IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ARLINGTON TECHNOLOGIES LLC,	§	
Plaintiff,	§	CIVIL ACTION NO. _________
v.	§	
RINGCENTRAL, INC.,	§	**JURY TRIAL DEMANDED**
Defendant.	§	

PLAINTIFF'S COMPLAINT FOR PATENT INFRINGEMENT

Plaintiff Arlington Technologies LLC ("ATL" or "Plaintiff") files this Complaint against Defendant RingCentral, Inc. ("Defendant" or "RingCentral") for infringement of U.S. Patent No. 7,366,110 (the "'110 Patent"), U.S. Patent No. 7,441,141 (the "'141 Patent"), U.S. Patent No. 8,145,945 (the "'945 Patent"), and U.S. Patent No. 9,026,836 (the "'836 Patent" and collectively, the "Asserted Patents").

THE PARTIES

1. Arlington Technologies LLC is a Texas limited liability company, with a principal place of business in Allen, TX.

2. Defendant RingCentral, Inc. is a corporation organized and existing under the laws of the State of Delaware, with a principal place of business at 20 Davis Drive, Belmont, California 94002. RingCentral, Inc. is registered to conduct business in the state of Delaware and has appointed The Corporation Trust Company, located at 1209 Orange Street, Wilmington, Delaware, 19801 as its agent for service of process.

3. On information and belief, Defendant is a multinational information technology company that develops and provides cloud-based communication and collaboration services, including phone, messaging, and video. Defendant sells its products and services to customers, including customers in this District.

4. On information and belief, Defendant operates and owns the RingCentral.com and Support.RingCentral.com websites, through which it markets, offers, distributes, and provides technical support for its cloud-based communication and collaboration services, including phone systems, messaging, and video, throughout the United States including in this District.

5. On information and belief, Defendant places, has placed, and/or contributed to placing Accused Products (as more particularly identified and described throughout this Complaint) into the stream of commerce via an established distribution channel knowing or understanding that such Accused Products would be sold in the United States, including in this District. Defendant has also derived substantial revenues from infringing acts in this District, including from the sale of the Accused Products.

6. On information and belief, Defendant is engaged in making, using, selling, offering for sale, and/or importing, and/or inducing its subsidiaries, affiliates, retail partners, and customers in the making, using, selling, offering for sale, and/or importing throughout the United States, including within this District, the products and services accused of infringement, such as cloud-based communication and collaboration services, including phone systems, messaging, and video.

7. Prior to the filing of the Complaint, Plaintiff attempted to engage Defendant and/or its agents in good faith licensing discussions related to the Asserted Patents, including by sending them correspondence on March 8, 2024, notifying Defendant of the need to license the Asserted Patents. Defendant's past and continuing sales of its devices (i) willfully infringe the Asserted

Patents and (ii) impermissibly take the significant benefits of Plaintiff's patented technologies without fair compensation to Plaintiff.

8. Through offers to sell, sales, imports, distributions, and other related agreements to transfer ownership of Defendant's products and/or services accused of infringement, such as cloud-based communication and collaboration services, including phone systems, messaging, and video, with distributors and customers operating in and maintaining a significant business presence in the U.S. and/or its U.S. subsidiaries Defendant does business in the U.S., the state of Delaware, and in this District.

<u>JURISDICTION AND VENUE</u>

9. This action arises under the patent laws of the United States, namely 35 U.S.C. §§ 271, 281, and 284-285, among others.

10. This Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

11. This Court has specific and general personal jurisdiction over Defendant consistent with the requirements of the Due Process Clause of the United States Constitution and the Delaware Long Arm Statute because, inter alia, (i) Defendant has engaged in continuous, systematic, and substantial business in Delaware; (ii) Defendant is registered to do business in Delaware; (iii) Defendant is incorporated in this District; (iv) Defendant has committed and continues to commit, acts of patent infringement in this State and in this District. Such acts of infringement include the making, using, testing, offering for sale, and selling of Accused Products (as more particularly identified and described throughout this Complaint, below) that leverage and infringe the inventions of the Asserted Patents in this State and this District and/or inducing others to commit acts of patent infringement in this State and District.

12. On information and belief, Defendant has purposefully and voluntarily placed, and is continuing to place, one or more Accused Products into the stream of commerce through established distribution channels (including the Internet) with the knowledge and intent that the Accused Products are and/or will be used, sold to and purchased by consumers in the United States, this State, and this District; and with the knowledge and expectation that the Accused Products (whether in standalone form or as integrated in downstream products) will be imported into the United States, this State, and this District.

13. Moreover, Defendant is organized under the laws of Delaware and incorporated in this District.

14. In addition, Defendant has derived substantial revenues from its infringing acts occurring within this State and this District. It has substantial business in this State and this District, including: (i) at least part of its infringing activities alleged herein; and (ii) regularly doing or soliciting business, engaging in other persistent conduct, and/or deriving substantial revenue from infringing goods offered for sale, sold, and imported, and services provided to Delaware residents. Defendant derives benefits from its presence in this District. For example, Defendant receives revenue from sales and distribution via electronic transactions conducted on and using at least its website located at www.RingCentral.com, and its incorporated and/or related systems.

15. In addition, Defendant has knowingly induced, and continues to knowingly induce, infringements within this State and this District by advertising, marketing, offering for sale and/or selling Accused Products (as more particularly identified and described throughout this Complaint) that incorporate the fundamental technologies covered by the Asserted Patents. Such advertising, marketing, offering for sale and/or selling of Accused Products is directed to consumers, customers, integrators, suppliers, distributors, resellers, partners, and/or end users, and this

includes providing instructions, user manuals, advertising, and/or marketing materials that facilitate, direct and encourage use of infringing functionality with Defendant's knowledge thereof.

16. Defendant has, thus, in the many ways described above, availed itself of the benefits and privileges of conducting business in this State and willingly subjected itself to the exercise of this Court's personal jurisdiction over it. Indeed, Defendant has sufficient minimum contacts with this forum through its transaction of substantial business in this State and this District and its commission of acts of patent infringement as alleged in this Complaint that are purposefully directed towards this State and District.

17. Venue is proper in this District pursuant to 28 U.S.C. §§ 1391 and 1400(b) because Defendant resides in this District under the Supreme Court's opinion in *TC Heartland v. Kraft Foods Group Brands LLC*, 137 S. Ct. 1514 (2017) through its incorporation in this District.

DEFENDANT'S PRE-SUIT KNOWLEDGE OF ITS INFRINGEMENTS

18. Prior to the filing of the Complaint, Plaintiff attempted to engage Defendant and/or its agents in good faith licensing discussions related to the Asserted Patents, including by sending them correspondence on March 8, 2024, notifying Defendant of the need to license the Asserted Patents. Defendant's past and continuing sales of its devices i) willfully infringe the Asserted Patents and ii) impermissibly take the significant benefits of Plaintiff's patented technologies without fair compensation to Plaintiff.

19. The Accused Products addressed in the Counts below include, but are not limited to, products and services identified in ATL's letter to Defendant. Defendant's past and continuing sales of the Accused Products (i) willfully infringe the Asserted Patents and (ii) impermissibly usurp the significant benefits of ATL's patented technologies without fair compensation.

THE ASSERTED PATENTS AND TECHNOLOGY

20. ATL is the sole and exclusive owner of all right, title, and interest in the Asserted Patents and holds the exclusive right to take all actions necessary to enforce its rights in, and to, the Asserted Patents, including the filing of this patent infringement lawsuit. Indeed, ATL owns all substantial rights in the Asserted Patents, including the right to exclude others and to recover damages for all past, present, and future infringements.

21. The ’110 Patent is entitled “Method and apparatus for merging call components during call reconstruction.” The ’110 Patent lawfully issued on April 29, 2008, and stems from U.S. Application No. 11/045,702, which was filed on January 27, 2005. A true and correct copy of the ’110 patent is attached hereto as Exhibit 1.

22. The ’141 Patent is entitled “Backup of network devices.” The ’141 Patent lawfully issued on October 21, 2008, and stems from U.S. Application No. 10/993,519, which was filed on November 22, 2004. A true and correct copy of the ’141 Patent is attached hereto as Exhibit 2.

23. The ’945 Patent is entitled “Packet mirroring between primary and secondary virtualized software images for improved system failover performance.” The ’945 Patent lawfully issued on March 27, 2021, and stems from U.S. Application No. 12/6.51,554, which was filed on January 4, 2010. A true and correct copy of the ’945 Patent is attached hereto as Exhibit 3.

24. The ’836 Patent is entitled “Call restoration in response to application failure.” The ’836 Patent lawfully issued on May 5, 2015, and stems from U.S. Application No. 13/476,789, which was filed on May 21, 2012. A true and correct copy of the ’836 Patent is attached hereto as Exhibit 4.

25. The claims of the Asserted Patents are directed to patent-eligible subject matter under 35 U.S.C. § 101. They are not directed to an abstract idea, and the technologies covered by

the claims comprise systems and/or ordered combinations of features and functions that, at the time of invention, were not, alone or in combination, well-understood, routine, or conventional.

26. To the extent necessary, ATL has complied with the requirements of 35 U.S.C. § 287, such that ATL may recover pre-suit damages.

COUNT I
(Infringement of U.S. Patent No. 7,366,110)

27. Plaintiff incorporates the preceding paragraphs herein by reference.

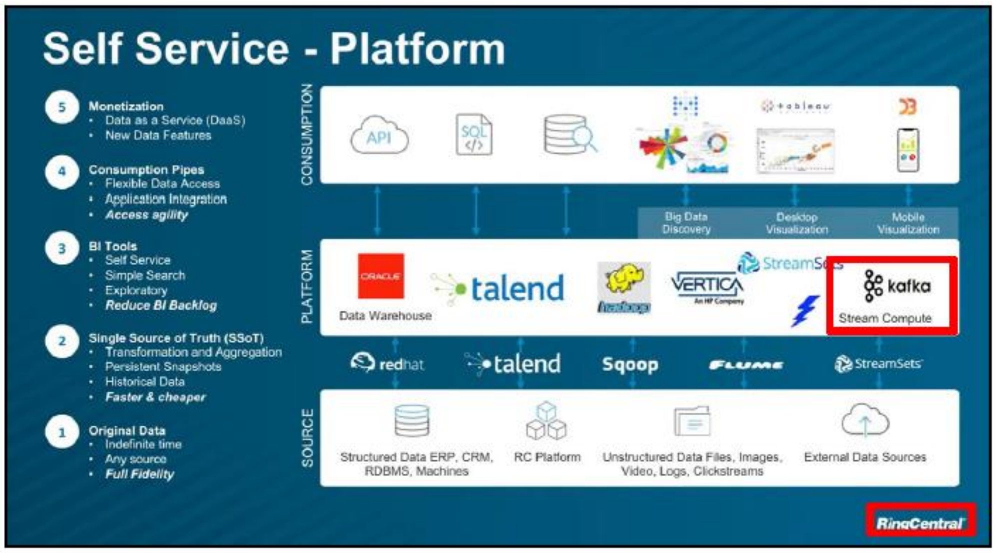
28. This cause of action arises under the patent laws of the United States, and, in particular, 35 U.S.C. §§ 271, *et seq*.

29. Plaintiff is the assignee of the ’110 Patent, with ownership of all substantial rights in the ’110 Patent, including the right to exclude others and to enforce, sue, and recover damages for past, present, and future infringements.

30. The ’110 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code after a full and fair examination.

31. Defendant has and continues to directly and/or indirectly infringe (by inducing infringement and/or contributing to infringement) one or more claims of the ’110 Patent in this District and elsewhere in Delaware and the United States.

32. Defendant designs, offers for sale, uses, and sells services, such as Apache Kafka (“the ’110 Accused Products”), in a manner that infringes the ’110 Patent. For example, Defendant uses Apache Kafka to support at least its Webinar and RingSense products:



Self Service - Platform

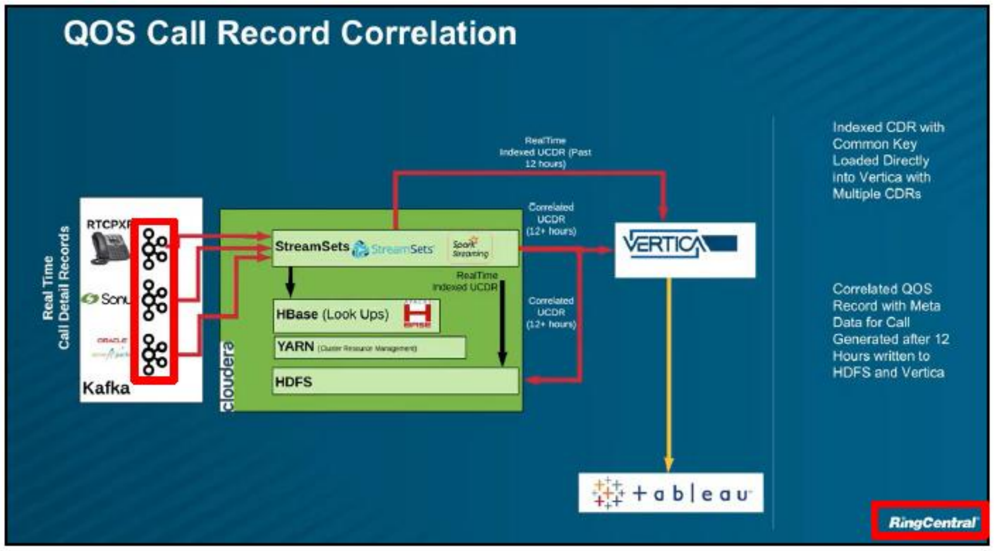
5 **Monetization**
- Data as a Service (DaaS)
- New Data Features

4 **Consumption Pipes**
- Flexible Data Access
- Application Integration
- ***Access agility***

3 **BI Tools**
- Self Service
- Simple Search
- Exploratory
- ***Reduce BI Backlog***

2 **Single Source of Truth (SSoT)**
- Transformation and Aggregation
- Persistent Snapshots
- Historical Data
- ***Faster & cheaper***

1 **Original Data**
- Indefinite time
- Any source
- ***Full Fidelity***

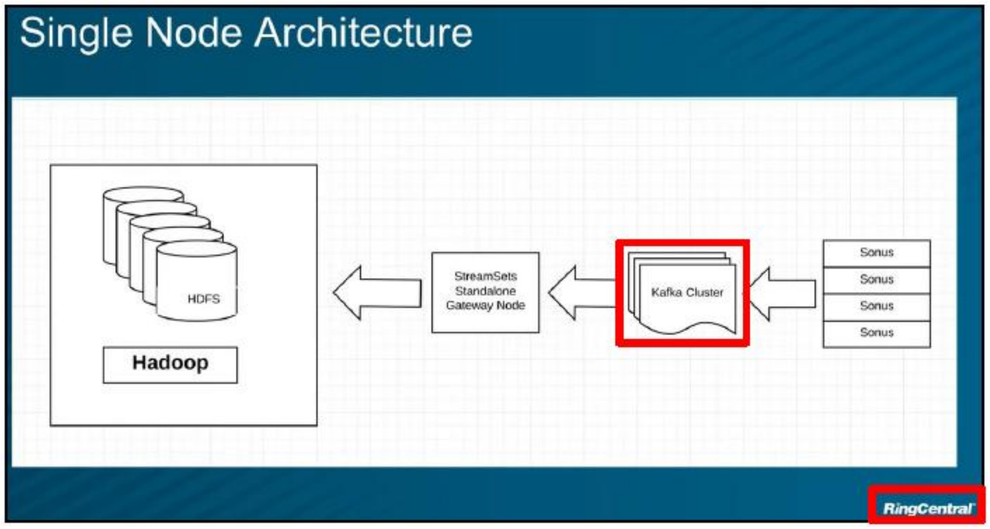


QOS Call Record Correlation

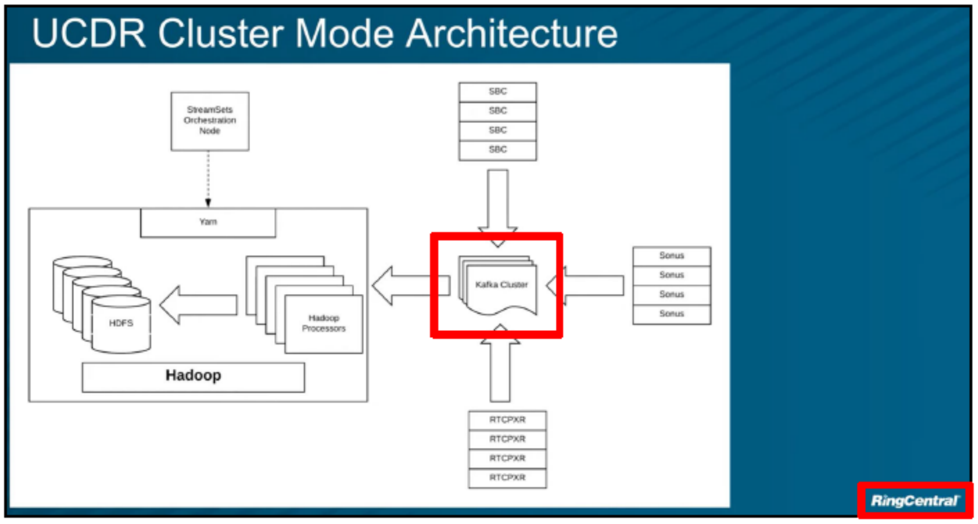
Real Time Call Detail Records
RTCPXR
Sonus
ORACLE
Kafka
cloudera
StreamSets
StreamSets
Spark Streaming
HBase (Look Ups)
YARN (Cluster Resource Management)
HDFS
RealTime Indexed UCDR (Past 12 hours)
RealTime Indexed UCDR
Correlated UCDR (12+ hours)
Correlated UCDR (12+ hours)
VERTICA
tableau
RingCentral

Indexed CDR with Common Key Loaded Directly into Vertica with Multiple CDRs

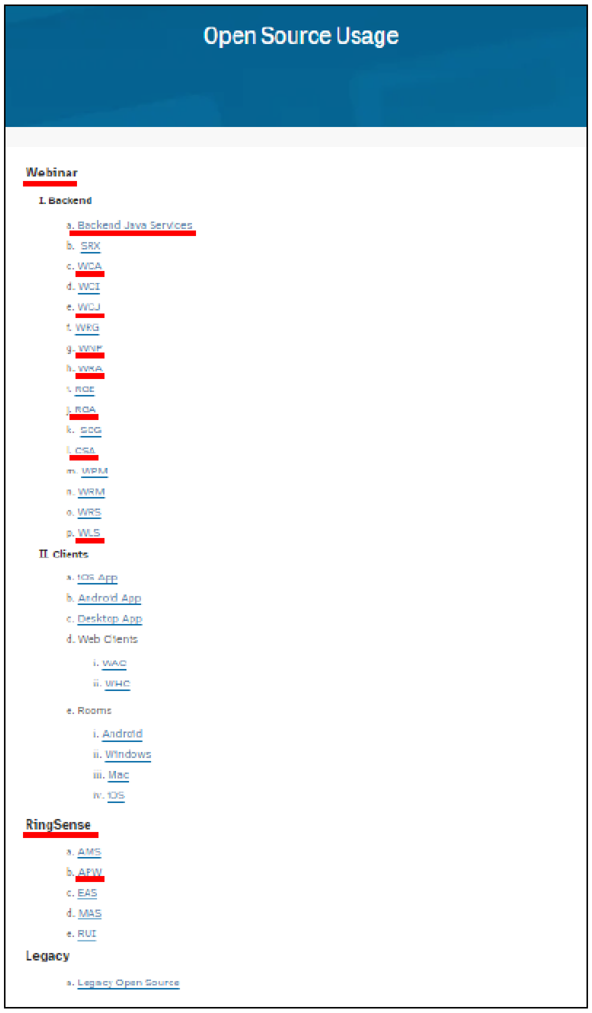
Correlated QOS Record with Meta Data for Call Generated after 12 Hours written to HDFS and Vertica



Single Node Architecture



Source: https://youtu.be/KMs-1BgQB0Q?si=tIlpYhp4jqc2d_Xi (emphasis added).



Open Source Usage

Webinar

I. Backend

- a. Backend Java Services
- b. SRX
- c. WCA
- d. WCI
- e. WCJ
- f. WRG
- g. WNF
- h. WRA
- i. RGE
- j. RGA
- k. SDG
- l. CSA
- m. WPM
- n. WRM
- o. WRS
- p. WLS

II. Clients

- a. iOS App
- b. Android App
- c. Desktop App
- d. Web Clients
  - i. WAC
  - ii. WHC
- e. Rooms
  - i. Android
  - ii. Windows
  - iii. Mac
  - iv. iOS

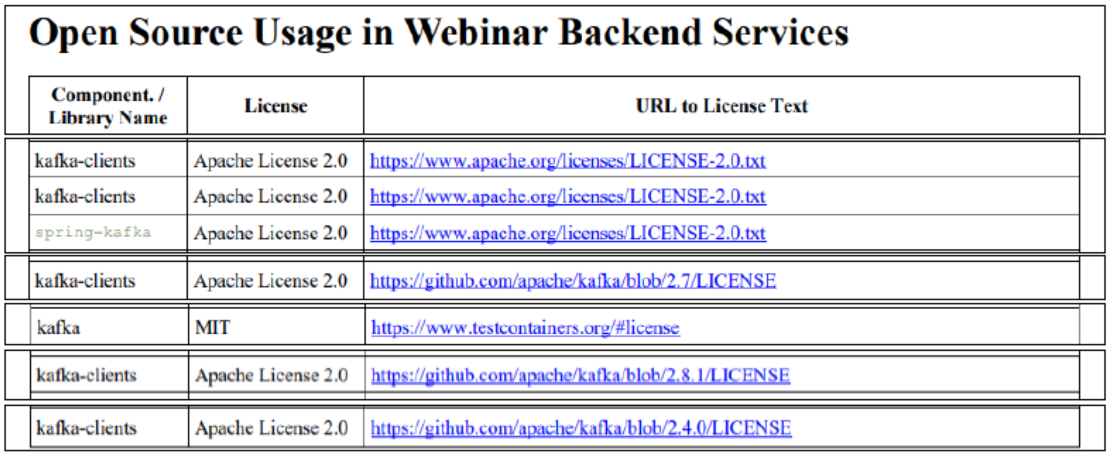
RingSense

- a. AMS
- b. APW
- c. EAS
- d. MAS
- e. RUI

Legacy

- a. Legacy Open Source

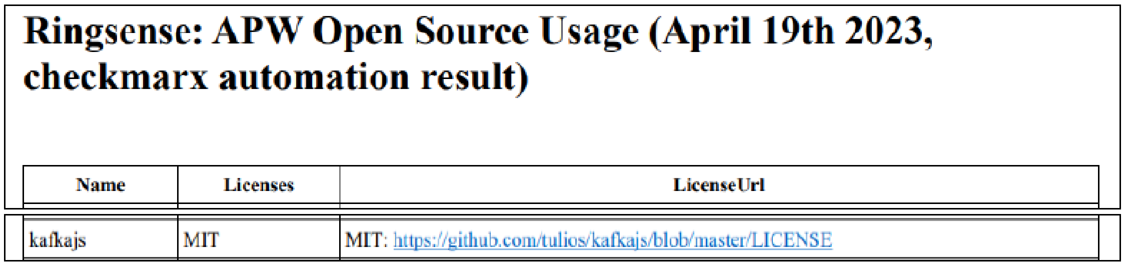
Source: https://www.ringcentral.com/legal/license_attribution.html (emphasis added).



Open Source Usage in Webinar Backend Services

Component. / Library Name	License	URL to License Text
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
spring-kafka	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.7/LICENSE
kafka	MIT	https://www.testcontainers.org/#license
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.8.1/LICENSE
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.4.0/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/backend-java-services.pdf.[1]



Ringsense: APW Open Source Usage (April 19th 2023, checkmarx automation result)

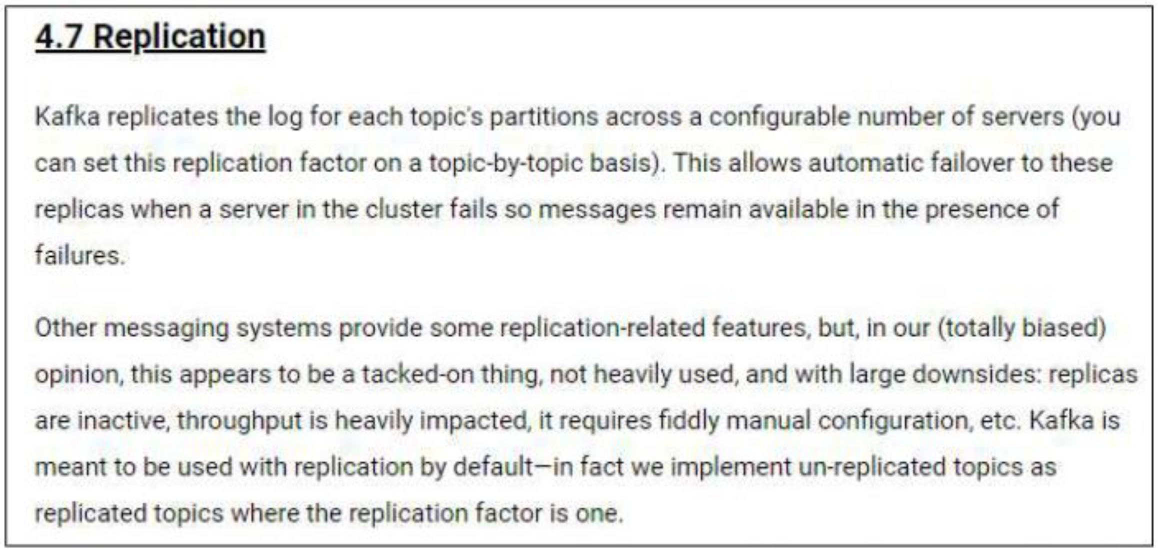
Name	Licenses	LicenseUrl
kafkajs	MIT	MIT: https://github.com/tulios/kafkajs/blob/master/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/apw.pdf.

33. Defendant directly infringes the ’110 Patent under 35 U.S.C. § 271(a) by using, making, offering for sale, selling, and/or importing the ’110 Accused Products, their components and processes, and/or products containing the same that incorporate the fundamental technologies covered by the ’110 Patent.

[1] *See also* https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wca.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wcj.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wnp.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wra.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/rga.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/csa.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wls-open-source-usage.pdf.

34. For example, Defendant infringes claim 1 of the ’110 Patent[2] via the ’110 Accused Products. The ’110 Accused Products are resilient to failure of a Kafka broker because they perform a method for migrating topic partitions from a first broker to a second broker:



4.7 Replication

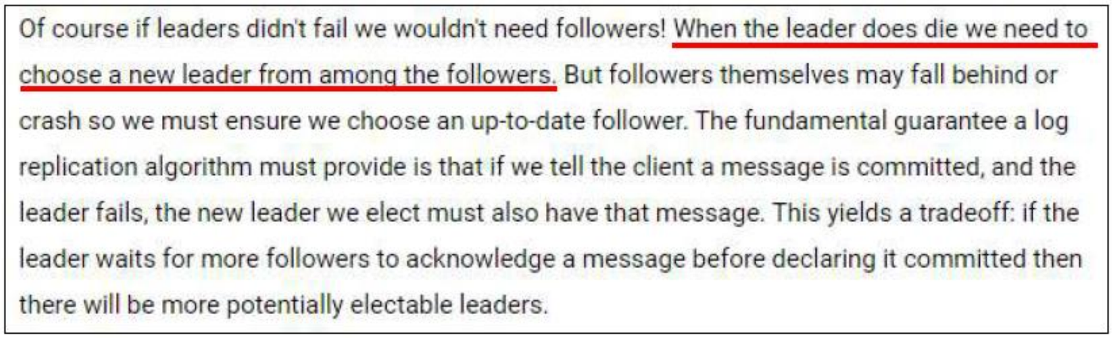
Kafka replicates the log for each topic's partitions across a configurable number of servers (you can set this replication factor on a topic-by-topic basis). This allows automatic failover to these replicas when a server in the cluster fails so messages remain available in the presence of failures.

Other messaging systems provide some replication-related features, but, in our (totally biased) opinion, this appears to be a tacked-on thing, not heavily used, and with large downsides: replicas are inactive, throughput is heavily impacted, it requires fiddly manual configuration, etc. Kafka is meant to be used with replication by default—in fact we implement un-replicated topics as replicated topics where the replication factor is one.

Source: https://kafka.apache.org/documentation/#replication.

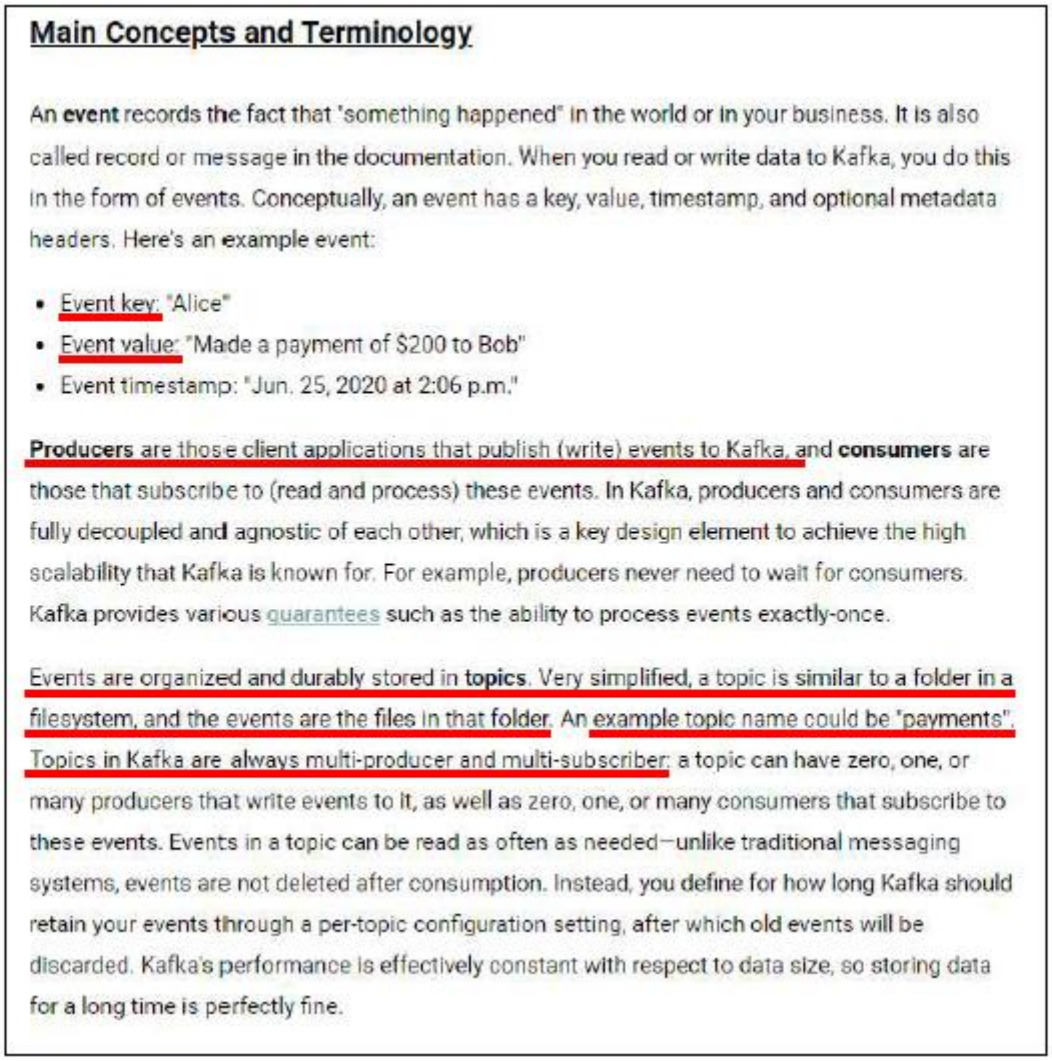
35. The ’110 Accused Products determine “that at least one communication is to be controlled by a second communication server, wherein the at least one communication was formerly controlled by a first communication server.” For example, when a leader broker fails, the Kafka cluster needs to elect a new leader for the partition that was controlled by the leader broker:

[2] Throughout this Complaint, wherever ATL identifies specific claims of the Asserted Patents infringed by Defendant, ATL expressly reserves the right to identify additional claims, products and/or services in its infringement contentions in accordance with applicable local rules and the Court’s case management order. Specifically identified claims throughout this Complaint are provided for notice pleading only.


Of course if leaders didn't fail we wouldn't need followers! When the leader does die we need to choose a new leader from among the followers. But followers themselves may fall behind or crash so we must ensure we choose an up-to-date follower. The fundamental guarantee a log replication algorithm must provide is that if we tell the client a message is committed, and the leader fails, the new leader we elect must also have that message. This yields a tradeoff: if the leader waits for more followers to acknowledge a message before declaring it committed then there will be more potentially electable leaders.

Source: https://kafka.apache.org/documentation/#replication (emphasis added).

36. The '110 Accused Products receive "from a first communication node, first communication information, wherein the first communication information is associated with the at least one communication and comprises at least one of a first node identifier and a communication identifier, the communication identifier is associated with the at least one communication, the second communication node comprises second communication information associated with the at least one communication and/or second node, and the first node identifier is associated with second communication information." For example, the Kafka cluster receives a first message from a first producer. The message includes a topic, a key, and a value. The key is a communication identifier:



Main Concepts and Terminology

An **event** records the fact that "something happened" in the world or in your business. It is also called record or message in the documentation. When you read or write data to Kafka, you do this in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers. Here's an example event:

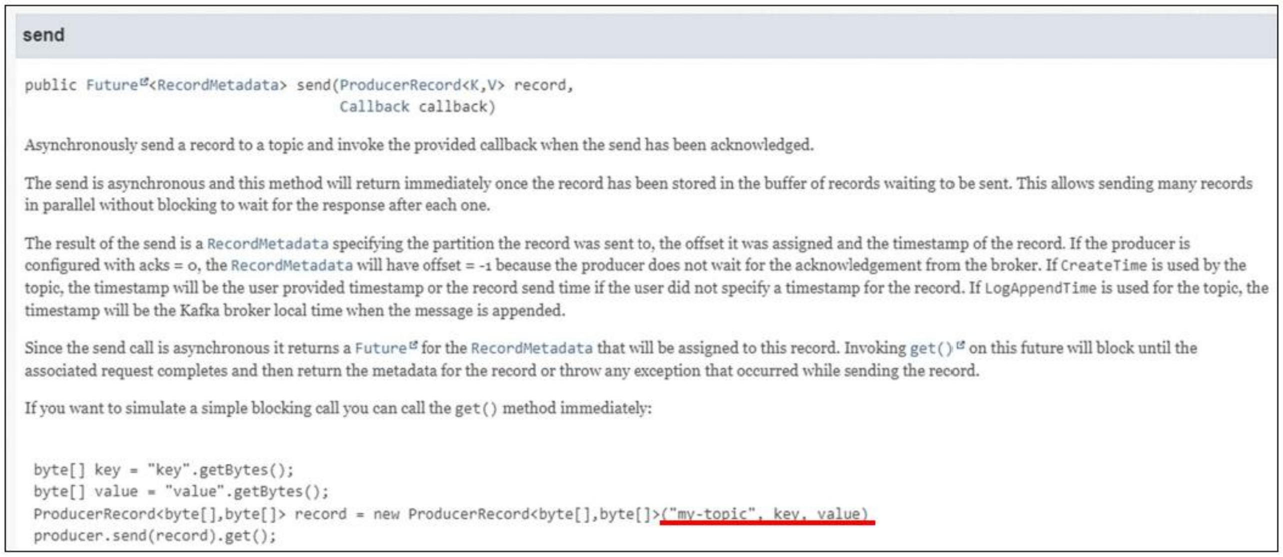
- Event key: "Alice"
- Event value: "Made a payment of $200 to Bob"
- Event timestamp: "Jun. 25, 2020 at 2:06 p.m."

Producers are those client applications that publish (write) events to Kafka, and **consumers** are those that subscribe to (read and process) these events. In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability that Kafka is known for. For example, producers never need to wait for consumers. Kafka provides various guarantees such as the ability to process events exactly-once.

Events are organized and durably stored in **topics**. Very simplified, a topic is similar to a folder in a filesystem, and the events are the files in that folder. An example topic name could be "payments". Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Events in a topic can be read as often as needed—unlike traditional messaging systems, events are not deleted after consumption. Instead, you define for how long Kafka should retain your events through a per-topic configuration setting, after which old events will be discarded. Kafka's performance is effectively constant with respect to data size, so storing data for a long time is perfectly fine.

Source: https://kafka.apache.org/documentation/#intro_concepts_and_terms (emphasis added).

Further, the key included in the message record is associated with a topic because the key determines which partition of a topic receives the message:


send

```
public Future<RecordMetadata> send(ProducerRecord<K,V> record,
                                   Callback callback)
```

Asynchronously send a record to a topic and invoke the provided callback when the send has been acknowledged.

The send is asynchronous and this method will return immediately once the record has been stored in the buffer of records waiting to be sent. This allows sending many records in parallel without blocking to wait for the response after each one.

The result of the send is a `RecordMetadata` specifying the partition the record was sent to, the offset it was assigned and the timestamp of the record. If the producer is configured with acks = 0, the `RecordMetadata` will have offset = -1 because the producer does not wait for the acknowledgement from the broker. If `CreateTime` is used by the topic, the timestamp will be the user provided timestamp or the record send time if the user did not specify a timestamp for the record. If `LogAppendTime` is used for the topic, the timestamp will be the Kafka broker local time when the message is appended.

Since the send call is asynchronous it returns a `Future` for the `RecordMetadata` that will be assigned to this record. Invoking `get()` on this future will block until the associated request completes and then return the metadata for the record or throw any exception that occurred while sending the record.

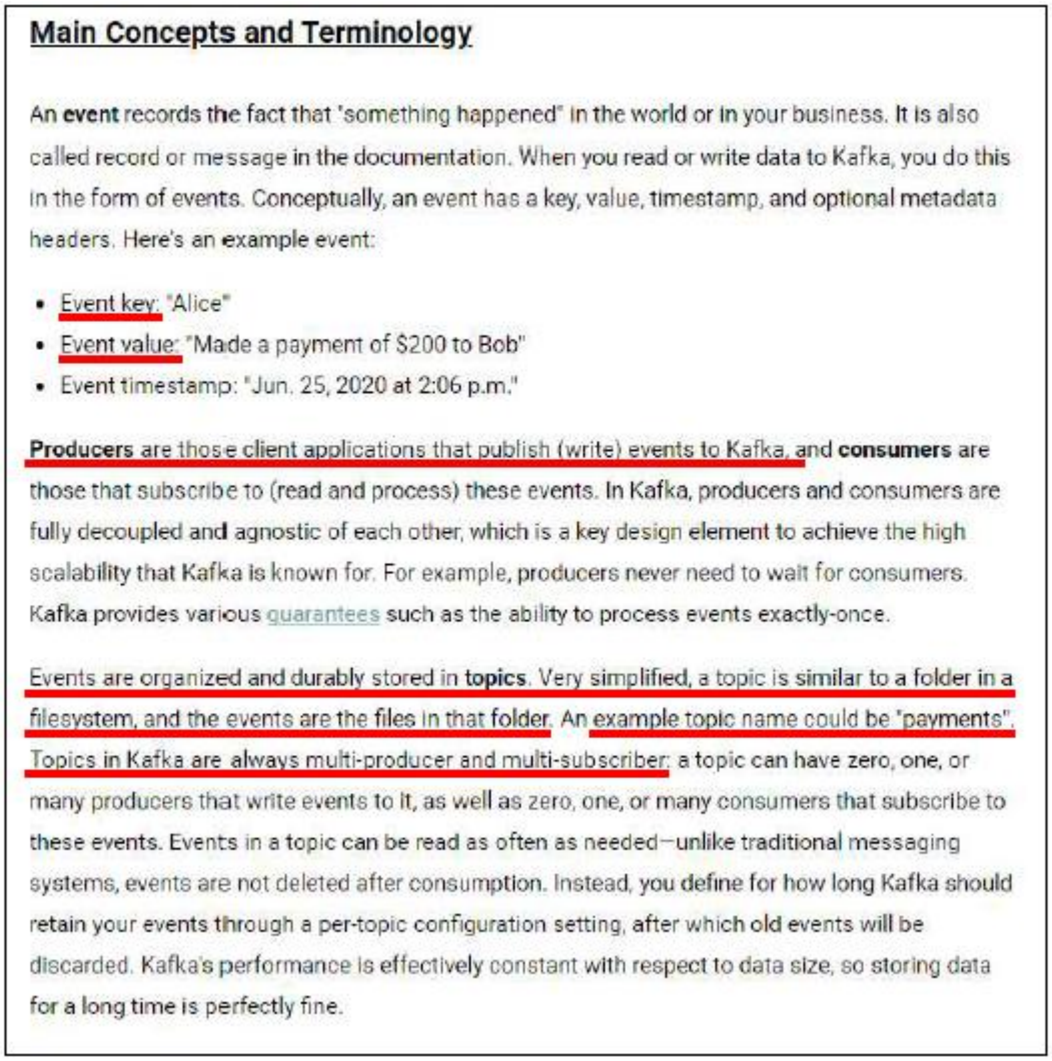
If you want to simulate a simple blocking call you can call the `get()` method immediately:

```
byte[] key = "key".getBytes();
byte[] value = "value".getBytes();
ProducerRecord<byte[],byte[]> record = new ProducerRecord<byte[],byte[]>("my-topic", key, value)
producer.send(record).get();
```

Source: https://kafka.apache.org/36/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html (emphasis added).

A second producer sends a second message to the Kafka cluster to be written to the same topic. The producer writes the second message, which comprises topic, key, and value information. As explained, the key included in the message record is associated with a topic because the key determines which partition of a topic receives the message. Because the key is associated with the topic, the key need not be associated with the second producer.

37. The ’110 Accused Products thereafter receive “from a second communication node, the second communication information.” For example, the Kafka cluster receives a second message from a second producer. The message includes a topic, a key, and a value:



Main Concepts and Terminology

An **event** records the fact that "something happened" in the world or in your business. It is also called record or message in the documentation. When you read or write data to Kafka, you do this in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers. Here's an example event:

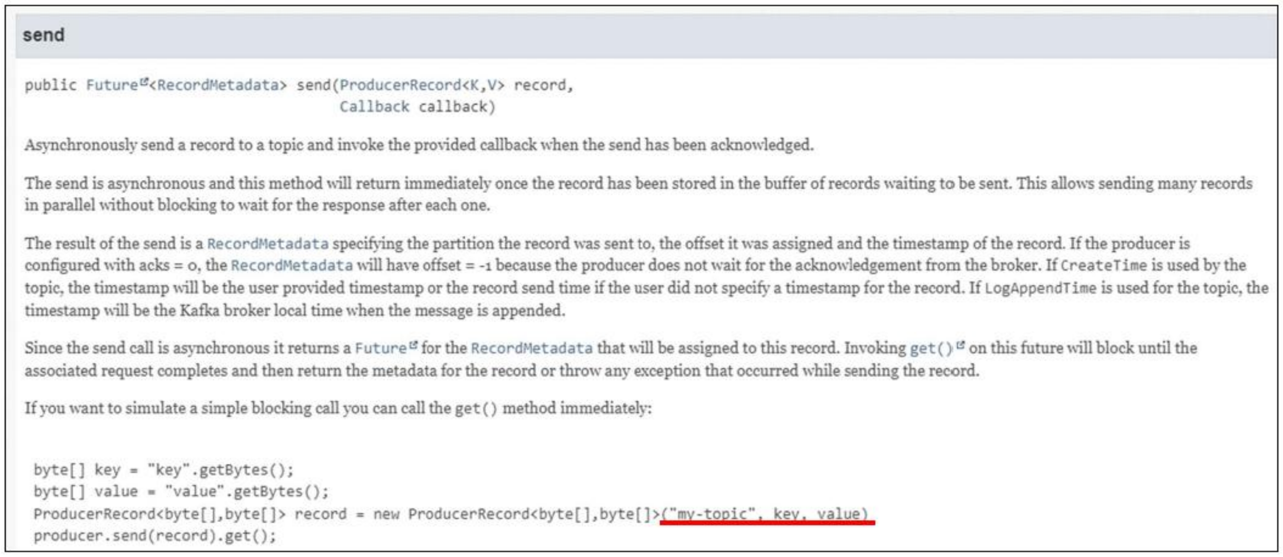
- Event key: "Alice"
- Event value: "Made a payment of $200 to Bob"
- Event timestamp: "Jun. 25, 2020 at 2:06 p.m."

Producers are those client applications that publish (write) events to Kafka, and **consumers** are those that subscribe to (read and process) these events. In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability that Kafka is known for. For example, producers never need to wait for consumers. Kafka provides various guarantees such as the ability to process events exactly-once.

Events are organized and durably stored in **topics**. Very simplified, a topic is similar to a folder in a filesystem, and the events are the files in that folder. An example topic name could be "payments". Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Events in a topic can be read as often as needed—unlike traditional messaging systems, events are not deleted after consumption. Instead, you define for how long Kafka should retain your events through a per-topic configuration setting, after which old events will be discarded. Kafka's performance is effectively constant with respect to data size, so storing data for a long time is perfectly fine.

Source: https://kafka.apache.org/documentation/#intro_concepts_and_terms (emphasis added).

The Kafka cluster receives a second message from a second producer. The message includes a topic, a key, and a value:



send

```
public Future<RecordMetadata> send(ProducerRecord<K,V> record,
                                    Callback callback)
```

Asynchronously send a record to a topic and invoke the provided callback when the send has been acknowledged.

The send is asynchronous and this method will return immediately once the record has been stored in the buffer of records waiting to be sent. This allows sending many records in parallel without blocking to wait for the response after each one.

The result of the send is a `RecordMetadata` specifying the partition the record was sent to, the offset it was assigned and the timestamp of the record. If the producer is configured with acks = 0, the `RecordMetadata` will have offset = -1 because the producer does not wait for the acknowledgement from the broker. If `CreateTime` is used by the topic, the timestamp will be the user provided timestamp or the record send time if the user did not specify a timestamp for the record. If `LogAppendTime` is used for the topic, the timestamp will be the Kafka broker local time when the message is appended.

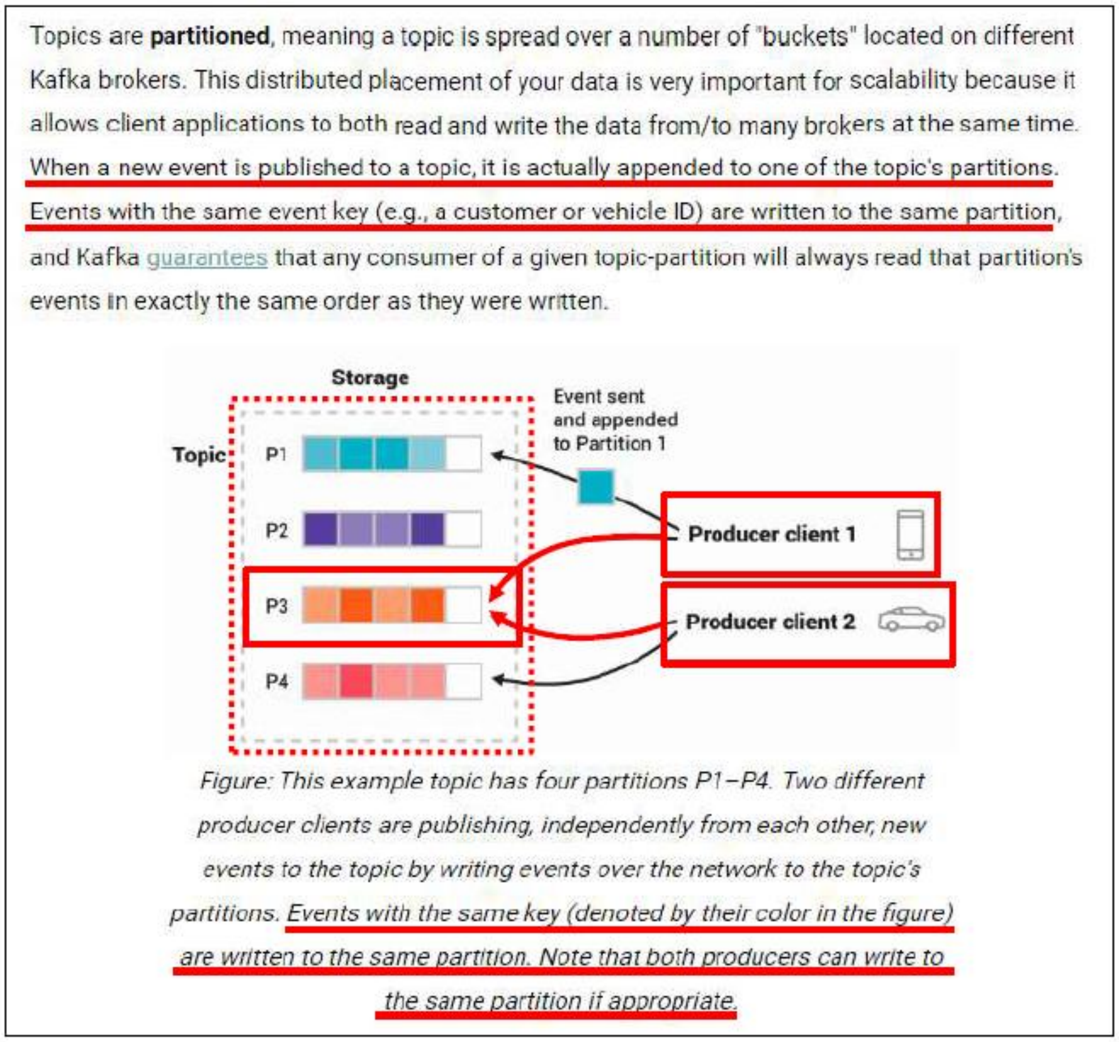
Since the send call is asynchronous it returns a `Future` for the `RecordMetadata` that will be assigned to this record. Invoking `get()` on this future will block until the associated request completes and then return the metadata for the record or throw any exception that occurred while sending the record.

If you want to simulate a simple blocking call you can call the `get()` method immediately:

```
byte[] key = "key".getBytes();
byte[] value = "value".getBytes();
ProducerRecord<byte[],byte[]> record = new ProducerRecord<byte[],byte[]>("my-topic", key, value)
producer.send(record).get();
```

Source: https://kafka.apache.org/36/javadoc/org/apache/kafka/clients/producer/KafkaProducer.html (emphasis added).

38. The '110 Accused Products identify "the second communication information based on the at least one of a first node identifier and communication identifier." For example, the second message is identified by the Kafka cluster based on the key. The key determines which topic partition the second message is appended to:



Topics are **partitioned**, meaning a topic is spread over a number of "buckets" located on different Kafka brokers. This distributed placement of your data is very important for scalability because it allows client applications to both read and write the data from/to many brokers at the same time. When a new event is published to a topic, it is actually appended to one of the topic's partitions. Events with the same event key (e.g., a customer or vehicle ID) are written to the same partition, and Kafka guarantees that any consumer of a given topic-partition will always read that partition's events in exactly the same order as they were written.

Figure: This example topic has four partitions P1–P4. Two different producer clients are publishing, independently from each other, new events to the topic by writing events over the network to the topic's partitions. Events with the same key (denoted by their color in the figure) are written to the same partition. Note that both producers can write to the same partition if appropriate.

Source: https://kafka.apache.org/documentation/#intro_concepts_and_terms (emphasis added).

39. The technology discussion above and the exemplary '110 Accused Products provide context for Plaintiff's infringement allegations.

40. At a minimum, Defendant has known of the '110 Patent at least as early as the filing date of the Complaint. In addition, Defendant has known about the '110 Patent since at least March 8, 2024, when Plaintiff sent correspondence to Defendant via FedEx alerting Defendant to its infringement.

41. On information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has actively induced, under U.S.C. § 271(b), its

distributors, customers, subsidiaries, importers, and/or consumers that import, purchase, or sell the ’110 Accused Products that include or are made using all of the limitations of one or more claims of the ’110 Patent to directly infringe one or more claims of the ’110 Patent (e.g., claim 1, as discussed above) by using, offering for sale, selling, and/or importing the ’110 Accused Products. Since at least the notice provided on the above-mentioned date, Defendant does so with knowledge, or with willful blindness of the fact, that the induced acts constitute infringement of the ’110 Patent. Defendant intends to cause, and has taken affirmative steps to induce infringement by its distributors, importers, customers, subsidiaries, and/or consumers by at least, *inter alia*, creating advertisements that promote the infringing use of the ’110 Accused Products, creating and/or maintaining established distribution channels for the ’110 Accused Products into and within the United States, manufacturing the ’110 Accused Products in conformity with U.S. laws and regulations, distributing or making available instructions or manuals for these products to purchasers and prospective buyers, and testing the ’110 Accused Products, and/or providing technical support, replacement parts, or services for these products to these purchasers in the United States. For example, Defendant configures Apache Kafka in the ’110 Accused Products to contain specific instructions, such as source code and configuration files, that cause such products to support at least its Webinar and RingSense products.

42. In the alternative, on information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has contributorily infringed, under U.S.C. § 271(c), one or more claims of the ’110 Patent. For example, Defendant contributes to the direct infringement of such claims by distributors, customers, subsidiaries, importers, and/or consumers that use, import, purchase, or sell the ’110 Accused Products. To the extent that the ’110 Accused Products do not directly infringe one or more claims of the ’110 Patent, such

products contain instructions, such as source code, that are especially adapted to cause the ’110 Accused Products to operate in an infringing manner. Such instructions are specifically designed to cause the ’110 Accused Products to provide and utilize Apache Kafka in an infringing manner and are a material part of the invention of the ’110 Patent and are not a staple article of commerce suitable for substantial non-infringing use.

43. On information and belief, despite having knowledge of the ’110 Patent and knowledge that it is directly and/or indirectly infringing one or more claims of the ’110 Patent, Defendant has nevertheless continued its infringing conduct and disregarded an objectively high likelihood of infringement. Defendant’s infringing activities relative to the ’110 Patent have been, and continue to be, willful, wanton, malicious, in bad-faith, deliberate, consciously wrongful, flagrant, and an egregious case of misconduct beyond typical infringement such that Plaintiff is entitled under 35 U.S.C. § 284 to enhanced damages up to three times the amount found or assessed.

44. Plaintiff has been damaged as a result of Defendant’s infringing conduct described in this Count. Defendant is, thus, liable to Plaintiff in an amount that adequately compensates Plaintiff for Defendant’s infringements, which, by law, cannot be less than a reasonable royalty, together with interest and costs as fixed by this Court under 35 U.S.C. § 284.

COUNT II
(Infringement of U.S. Patent No. 7,441,141)

45. Plaintiff incorporates the preceding paragraphs herein by reference.

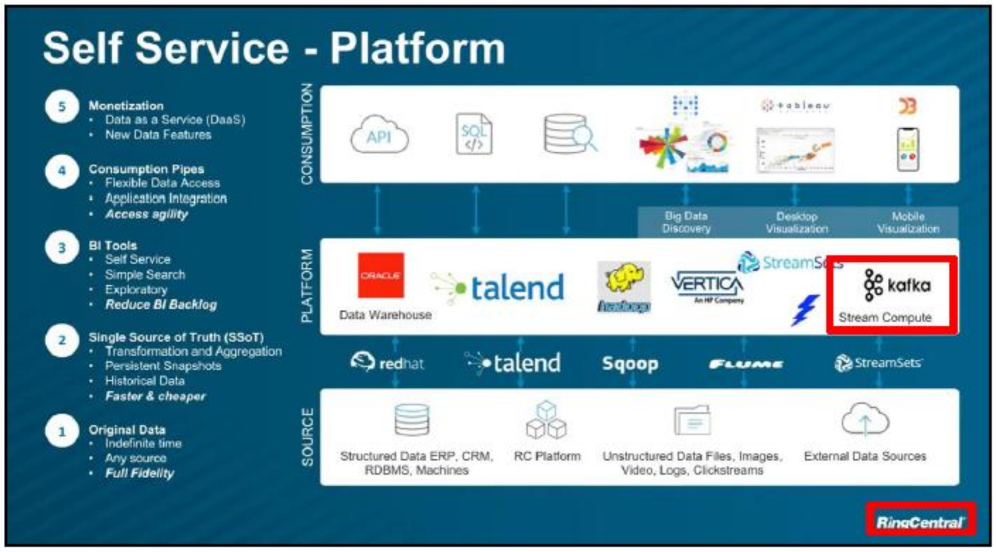
46. This cause of action arises under the patent laws of the United States, and, in particular, 35 U.S.C. §§ 271, *et seq*.

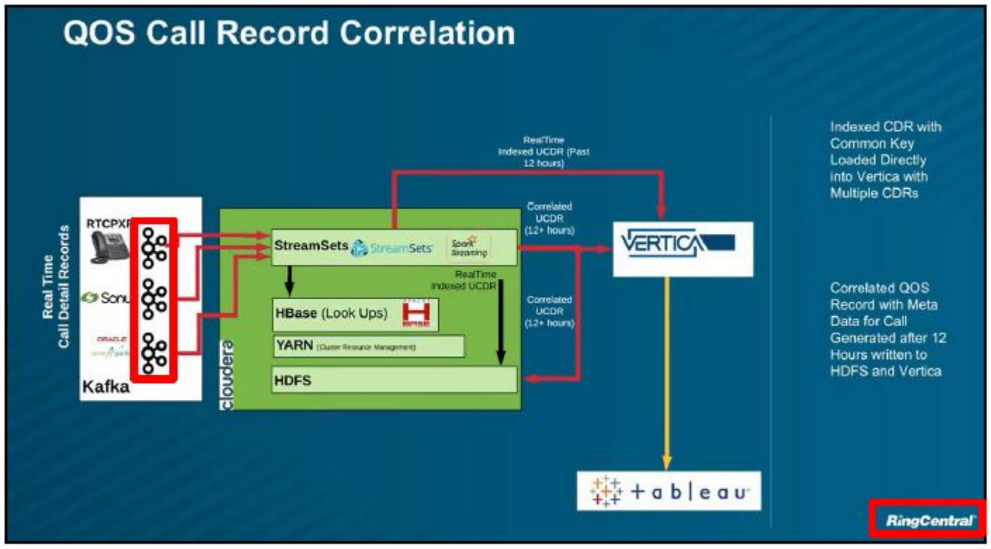
47. Plaintiff is the assignee of the ’141 Patent, with ownership of all substantial rights in the ’141 Patent, including the right to exclude others and to enforce, sue, and recover damages for past, present, and future infringements.

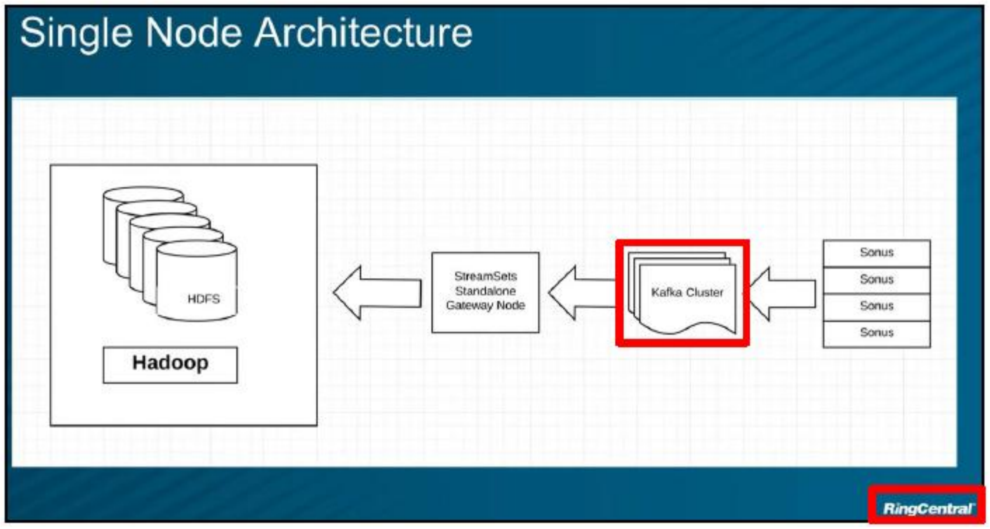
48. The ’141 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code after a full and fair examination.

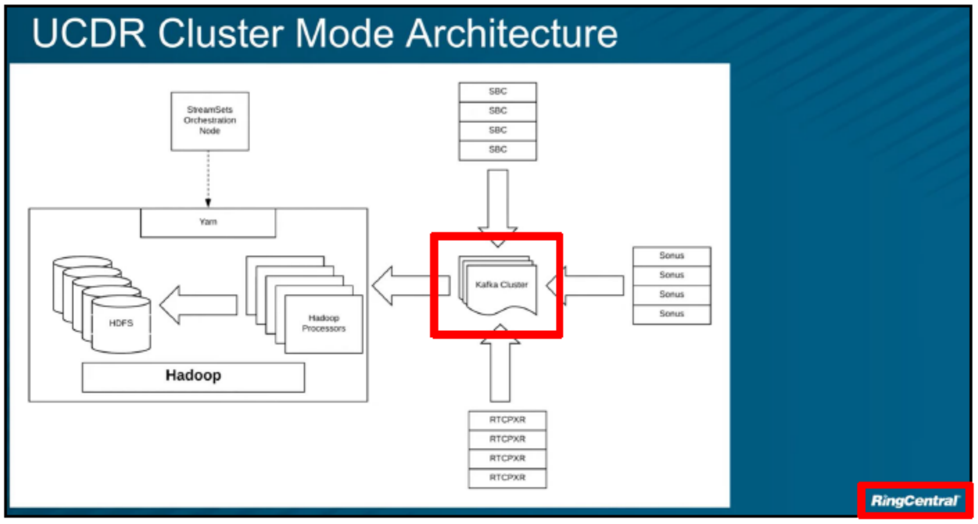
49. Defendant has and continues to directly and/or indirectly infringe (by inducing infringement and/or contributing to infringement) one or more claims of the ’141 Patent in this judicial district and elsewhere in Delaware and the United States.

50. Defendant designs, offers for sale, uses, and sells services, such as Apache Kafka (“the ’141 Accused Products”), in a manner that infringes the ’141 Patent. For example, Defendant uses Apache Kafka to support at least its Webinar and RingSense products:

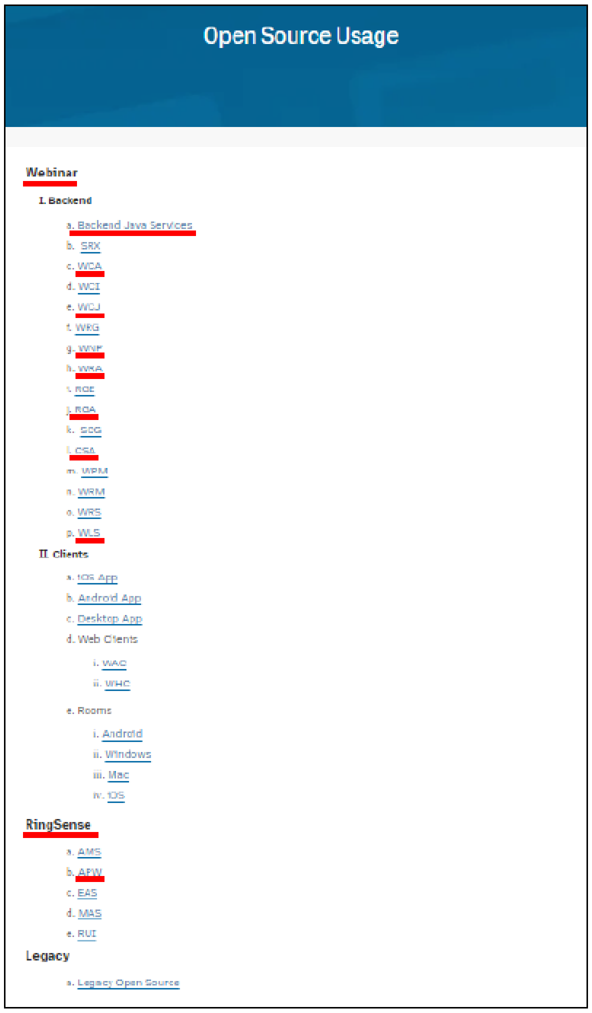



QOS Call Record Correlation
Real Time Call Detail Records
RTCPXR
Sonus
Kafka
cloudera
StreamSets
Spark Streaming
HBase (Look Ups)
YARN (Cluster Resource Management)
HDFS
RealTime Indexed UCDR (Past 12 hours)
Correlated UCDR (12+ hours)
RealTime Indexed UCDR
Correlated UCDR (12+ hours)
VERTICA
tableau
Indexed CDR with Common Key Loaded Directly into Vertica with Multiple CDRs
Correlated QOS Record with Meta Data for Call Generated after 12 Hours written to HDFS and Vertica
RingCentral





Source: https://youtu.be/KMs-1BgQB0Q?si=tIlpYhp4jqc2d_Xi (emphasis added).



Open Source Usage

Webinar

I. Backend

- a. Backend Java Services
- b. SRX
- c. WCA
- d. WCI
- e. WCJ
- f. WRG
- g. WNF
- h. WRA
- i. RGE
- j. RGA
- k. SGG
- l. CSA
- m. WPM
- n. WRM
- o. WRS
- p. WLS

II. Clients

- a. iOS App
- b. Android App
- c. Desktop App
- d. Web Clients
  - i. WAC
  - ii. WHC
- e. Rooms
  - i. Android
  - ii. Windows
  - iii. Mac
  - iv. iOS

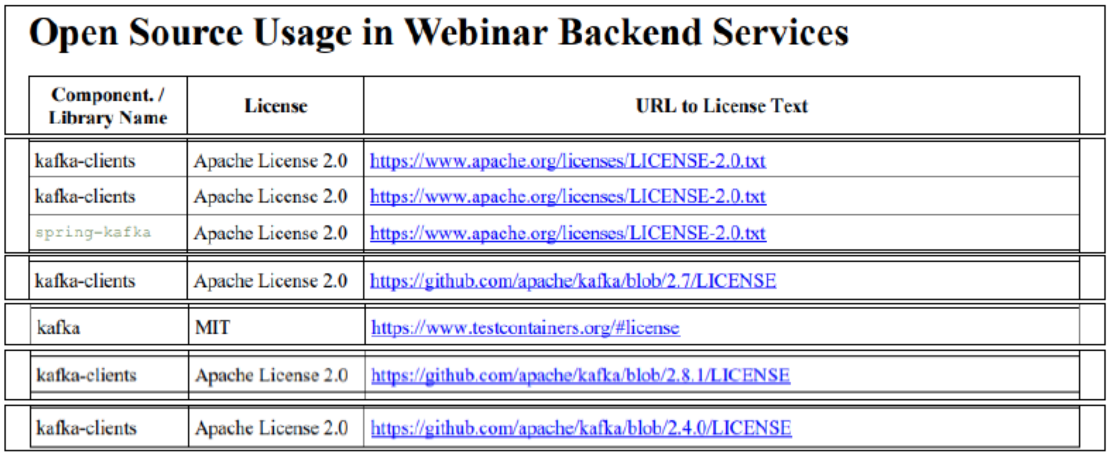
RingSense

- a. AMS
- b. APW
- c. EAS
- d. MAS
- e. RUI

Legacy

- a. Legacy Open Source

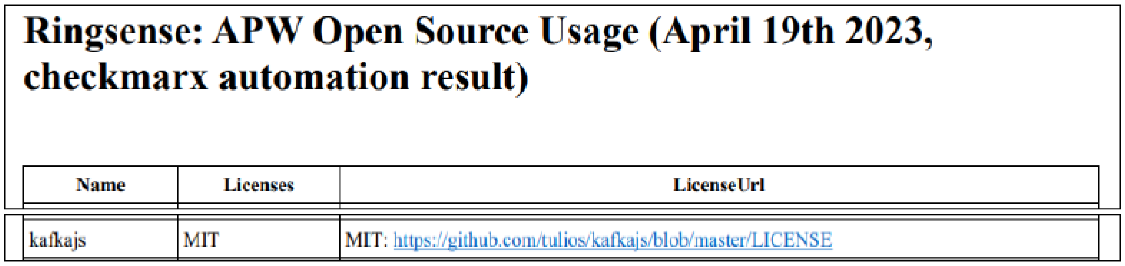
Source: https://www.ringcentral.com/legal/license_attribution.html (emphasis added).



Open Source Usage in Webinar Backend Services

Component. / Library Name	License	URL to License Text
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
spring-kafka	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.7/LICENSE
kafka	MIT	https://www.testcontainers.org/#license
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.8.1/LICENSE
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.4.0/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/backend-java-services.pdf.[3]


Ringsense: APW Open Source Usage (April 19th 2023, checkmarx automation result)

Name	Licenses	LicenseUrl
kafkajs	MIT	MIT: https://github.com/tulios/kafkajs/blob/master/LICENSE

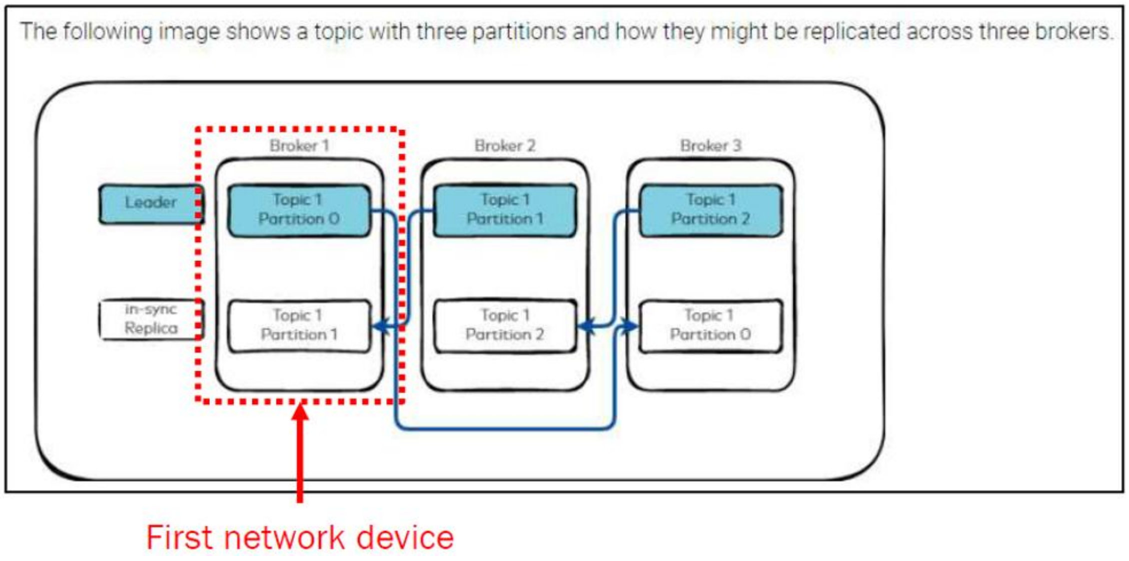
Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/apw.pdf.

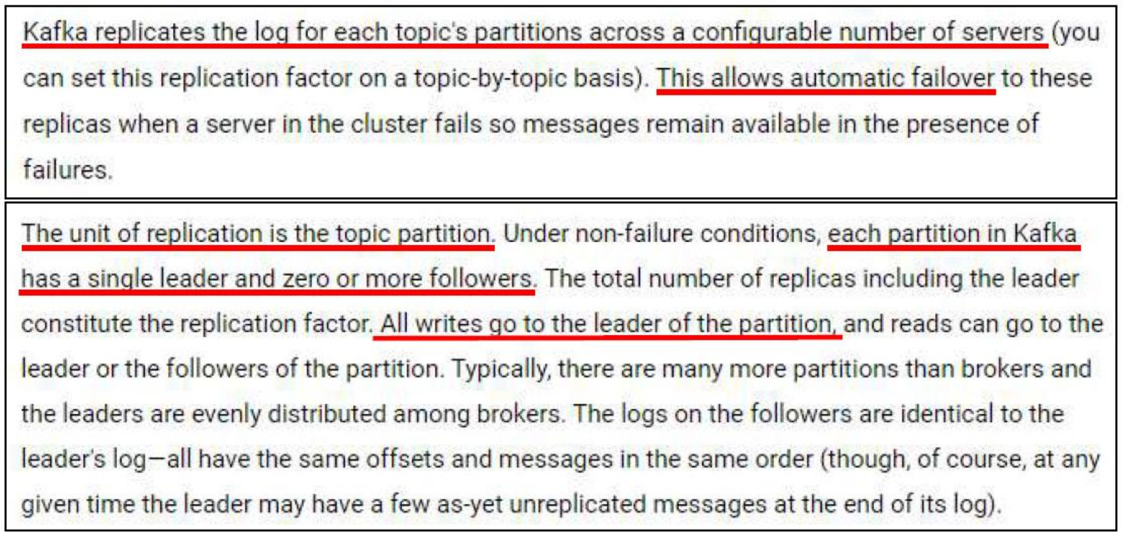
51. Defendant directly infringes the ’141 Patent under 35 U.S.C. § 271(a) by using, making, offering for sale, selling, and/or importing the ’141 Accused Products, their components and processes, and/or products containing the same that incorporate the fundamental technologies covered by the ’141 Patent.

52. For example, Defendant infringes claim 1 of the ’141 Patent via the ’141 Accused Products. The ’141 Accused Products operate at “a first network device of a plurality of network devices each storing device-specific information.” For example, Kafka broker servers are network devices that store device-specific information (i.e., partitions for which they are leaders).

[3] *See also* https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wca.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wcj.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wnp.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wra.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/rga.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/csa.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wls-open-source-usage.pdf.

The following image shows a topic with three partitions and how they might be replicated across three brokers.



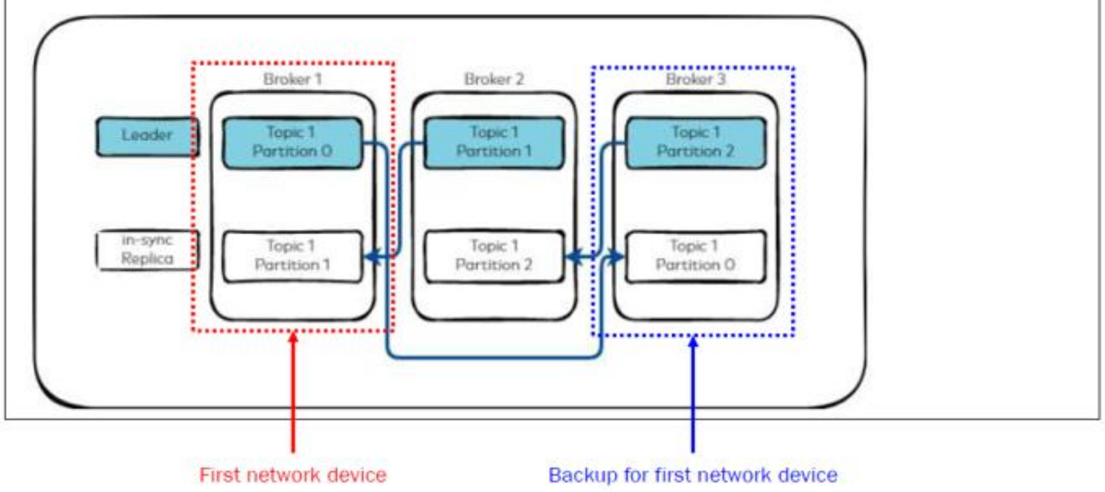

Kafka replicates the log for each topic's partitions across a configurable number of servers (you can set this replication factor on a topic-by-topic basis). This allows automatic failover to these replicas when a server in the cluster fails so messages remain available in the presence of failures.

The unit of replication is the topic partition. Under non-failure conditions, each partition in Kafka has a single leader and zero or more followers. The total number of replicas including the leader constitute the replication factor. All writes go to the leader of the partition, and reads can go to the leader or the followers of the partition. Typically, there are many more partitions than brokers and the leaders are evenly distributed among brokers. The logs on the followers are identical to the leader's log—all have the same offsets and messages in the same order (though, of course, at any given time the leader may have a few as-yet unreplicated messages at the end of its log).

Source: https://kafka.apache.org/documentation/#replication (emphasis added).

53. The ’141 Accused Products select “at least one second network device of said plurality of network devices to act as a backup for said first network device.” For example, a leader broker selects a second broker to act as a backup:

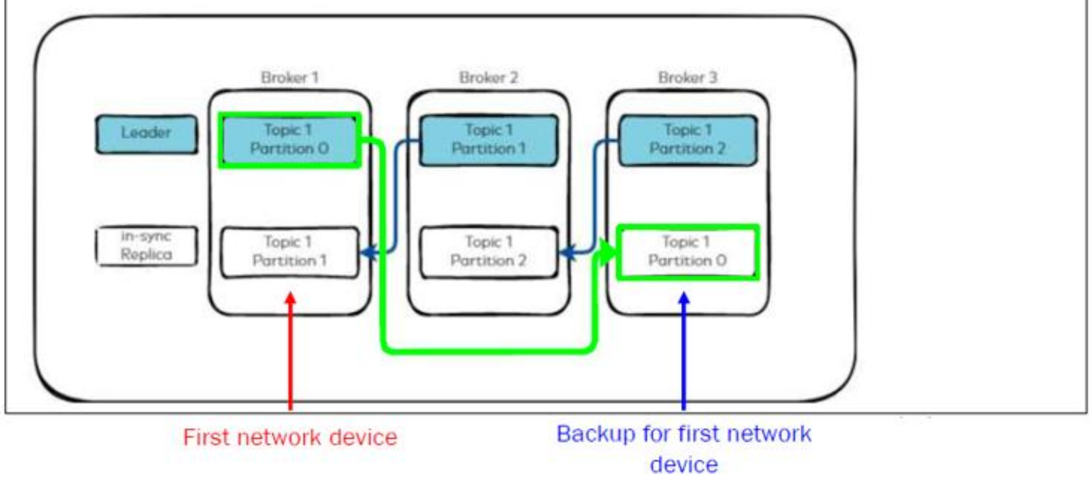
The following image shows a topic with three partitions and how they might be replicated across three brokers.



Source: https://docs.confluent.io/kafka/design/replication.html (emphasis added).

54. The '141 Accused Products communicate "the device-specific information maintained by said first network device to said at least one second network device, said communicated device-specific information for use by said at least one second network device in assuming the role of said first network device upon unavailability of said first network device." For example, the Kafka leader broker communicates message records (e.g., Partition 0) to the backup broker so that the backup broker can assume the role of leader if the leader broker is unavailable:

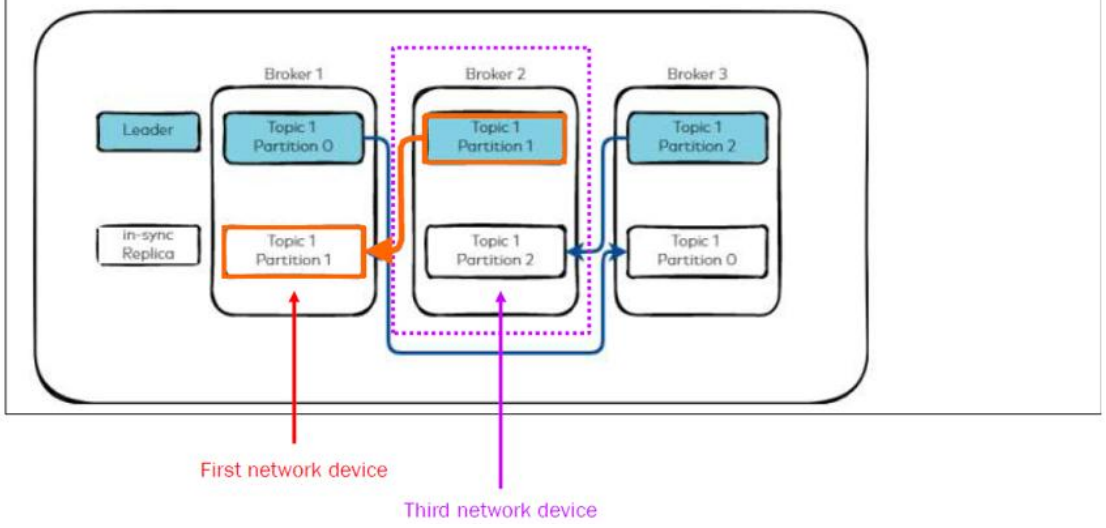
The following image shows a topic with three partitions and how they might be replicated across three brokers.



Source: https://docs.confluent.io/kafka/design/replication.html (emphasis added).

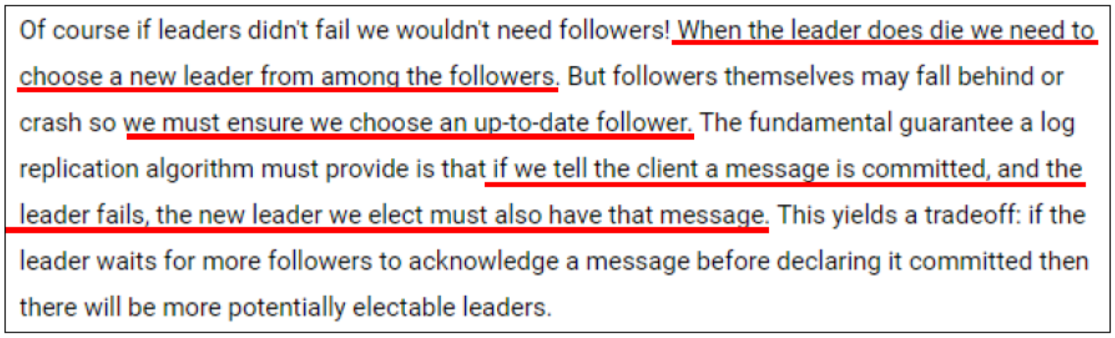
55. The ’141 Accused Products thereafter receive “at said first network device device-specific information from at least one third network device for use by said first network device in assuming the role of the third network device upon unavailability of the third network device.” For example, the leader broker may be a follower broker for a different topic partition (e.g., Partition 1):

The following image shows a topic with three partitions and how they might be replicated across three brokers.

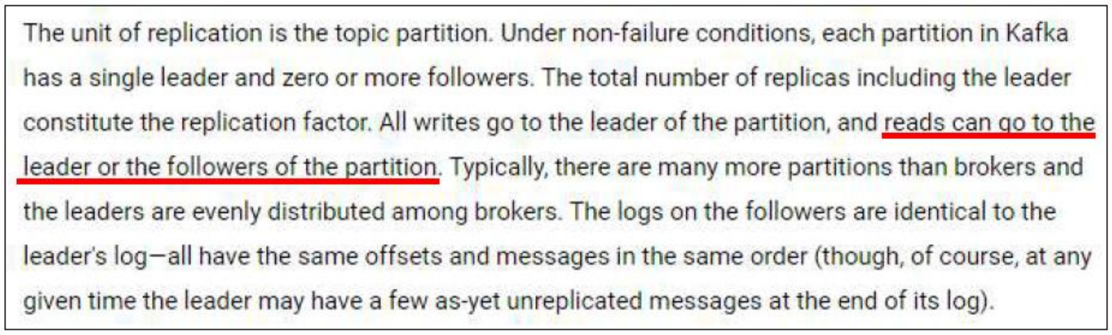


Source: https://docs.confluent.io/kafka/design/replication.html (emphasis added).

56. The ’141 Accused Products operate such that “when the device-specific information of said first network device is requested and said first network device is unavailable, communicating the device-specific information of said first network device from one of said at least one second network device.” For example, when messages stored in the topic partition of the leader broker is requested and the leader broker is unavailable, the messages of the topic partition are consumed from the follower broker.


Of course if leaders didn't fail we wouldn't need followers! When the leader does die we need to choose a new leader from among the followers. But followers themselves may fall behind or crash so we must ensure we choose an up-to-date follower. The fundamental guarantee a log replication algorithm must provide is that if we tell the client a message is committed, and the leader fails, the new leader we elect must also have that message. This yields a tradeoff: if the leader waits for more followers to acknowledge a message before declaring it committed then there will be more potentially electable leaders.

Source: https://kafka.apache.org/documentation/#design_replicatedlog (emphasis added).


The unit of replication is the topic partition. Under non-failure conditions, each partition in Kafka has a single leader and zero or more followers. The total number of replicas including the leader constitute the replication factor. All writes go to the leader of the partition, and reads can go to the leader or the followers of the partition. Typically, there are many more partitions than brokers and the leaders are evenly distributed among brokers. The logs on the followers are identical to the leader's log—all have the same offsets and messages in the same order (though, of course, at any given time the leader may have a few as-yet unreplicated messages at the end of its log).

Source: https://kafka.apache.org/documentation/#replication (emphasis added).

57. The technology discussion above and the exemplary ’141 Accused Products provide context for Plaintiff’s infringement allegations.

58. At a minimum, Defendant has known of the ’141 Patent at least as early as the filing date of the Complaint. In addition, Defendant has known about the ’141 Patent since at least March 8, 2024, when Plaintiff sent correspondence to Defendant via FedEx alerting Defendant to its infringement.

59. On information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has actively induced, under U.S.C. § 271(b), its distributors, customers, subsidiaries, importers, and/or consumers that import, purchase, or sell the ’141 Accused Products that include or are made using all of the limitations of one or more claims of the ’141 patent to directly infringe one or more claims of the ’141 Patent (e.g., claim 1, as discussed above) by using, offering for sale, selling, and/or importing the ’141 Accused Products.

Since at least the notice provided on the above-mentioned date, Defendant does so with knowledge, or with willful blindness of the fact, that the induced acts constitute infringement of the ’141 Patent. Defendant intends to cause, and has taken affirmative steps to induce infringement by its distributors, importers, customers, subsidiaries, and/or consumers by at least, *inter alia*, creating advertisements that promote the infringing use of the ’141 Accused Products, creating and/or maintaining established distribution channels for the ’141 Accused Products into and within the United States, manufacturing the ’141 Accused Products in conformity with U.S. laws and regulations, distributing or making available instructions or manuals for these products to purchasers and prospective buyers, and testing the ’141 Accused Products, and/or providing technical support, replacement parts, or services for these products to these purchasers in the United States. For example, Defendant configures Apache Kafka in the ’141 Accused Products to contain specific instructions, such as source code and configuration files, that cause such products to support at least its Webinar and RingSense products.

60. In the alternative, on information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has contributorily infringed, under U.S.C. § 271(c), one or more claims of the ’141 Patent. For example, Defendant contributes to the direct infringement of such claims by distributors, customers, subsidiaries, importers, and/or consumers that use, import, purchase, or sell the ’141 Accused Products. To the extent that the ’141 Accused Products do not directly infringe one or more claims of the ’141 patent, such products contain instructions, such as source code, that are especially adapted to cause the ’141 Accused Products to operate in an infringing manner. Such instructions are specifically designed to cause the ’141 Accused Products to provide and utilize Apache Kafka in an infringing manner

and are a material part of the invention of the ’141 Patent and are not a staple article of commerce suitable for substantial non-infringing use.

61. On information and belief, despite having knowledge of the ’141 Patent and knowledge that it is directly and/or indirectly infringing one or more claims of the ’141 Patent, Defendant has nevertheless continued its infringing conduct and disregarded an objectively high likelihood of infringement. Defendant’s infringing activities relative to the ’141 Patent have been, and continue to be, willful, wanton, malicious, in bad-faith, deliberate, consciously wrongful, flagrant, and an egregious case of misconduct beyond typical infringement such that Plaintiff is entitled under 35 U.S.C. § 284 to enhanced damages up to three times the amount found or assessed.

62. Plaintiff has been damaged as a result of Defendant’s infringing conduct described in this Count. Defendant is, thus, liable to Plaintiff in an amount that adequately compensates Plaintiff for Defendant’s infringements, which, by law, cannot be less than a reasonable royalty, together with interest and costs as fixed by this Court under 35 U.S.C. § 284.

COUNT III
(Infringement of U.S. Patent No. 8,145,945)

63. Plaintiff incorporates the preceding paragraphs herein by reference.

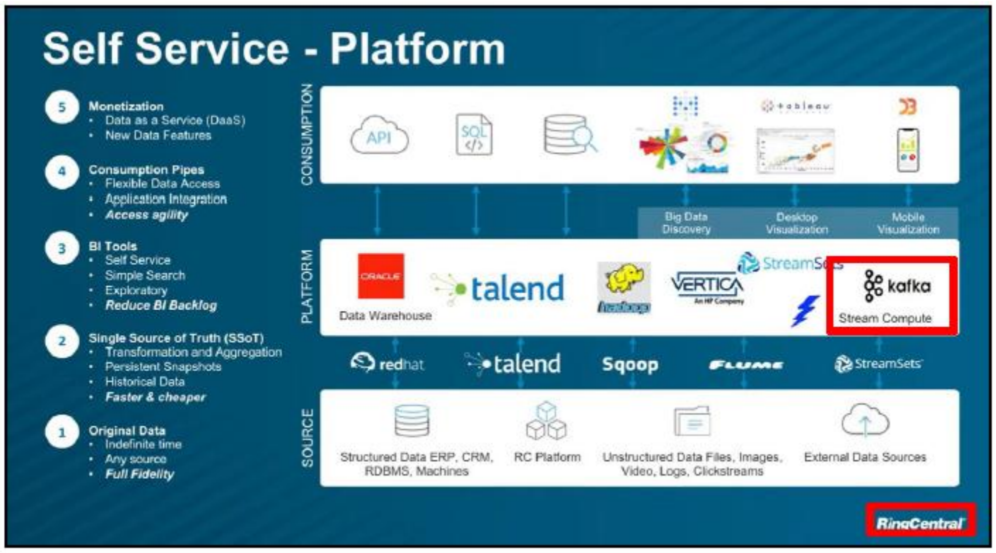
64. This cause of action arises under the patent laws of the United States, and, in particular, 35 U.S.C. §§ 271, *et seq*.

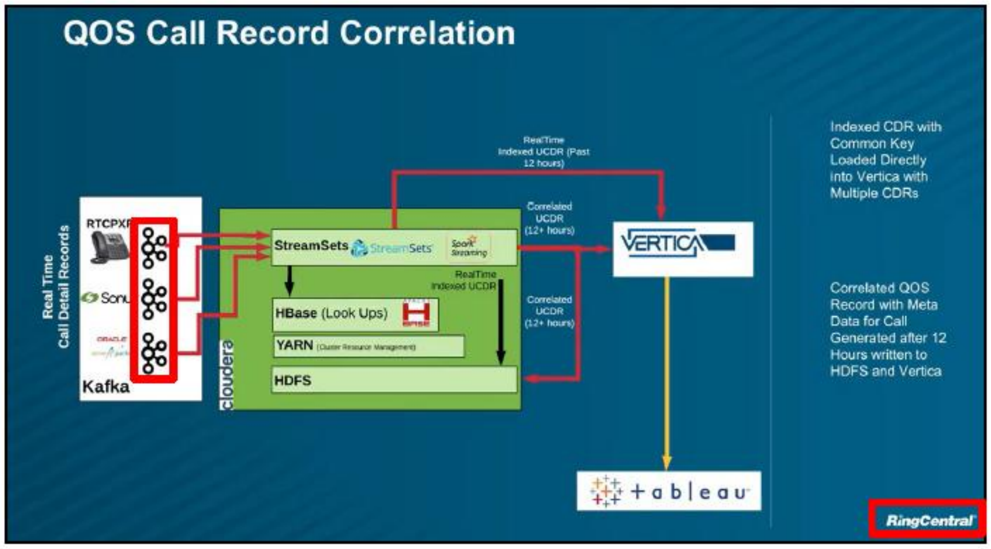
65. Plaintiff is the assignee of the ’945 Patent, with ownership of all substantial rights in the ’945 Patent, including the right to exclude others and to enforce, sue, and recover damages for past, present, and future infringements.

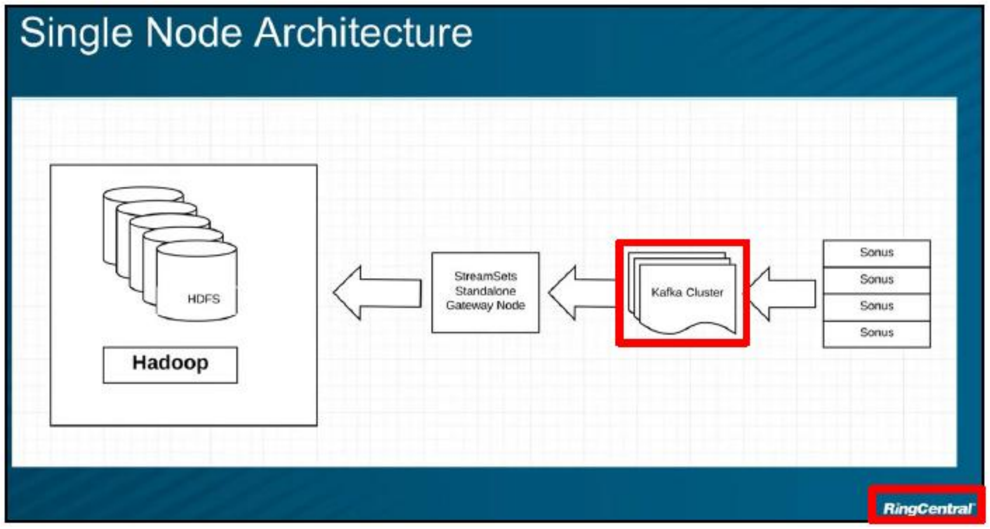
66. The ’945 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code after a full and fair examination.

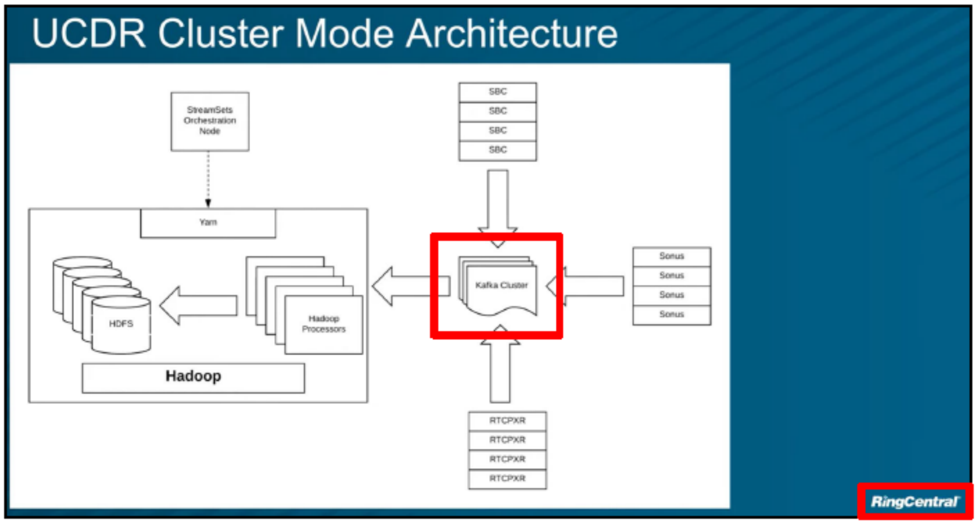
67. Defendant has and continues to directly and/or indirectly infringe (by inducing infringement and/or contributing to infringement) one or more claims of the ’945 Patent in this judicial district and elsewhere in Delaware and the United States.

68. Defendant designs, offers for sale, uses, and sells services, such as Apache Kafka (“the ’945 Accused Products”), in a manner that infringes the ’945 Patent. For example, Defendant uses Apache Kafka to support at least its Webinar and RingSense products:

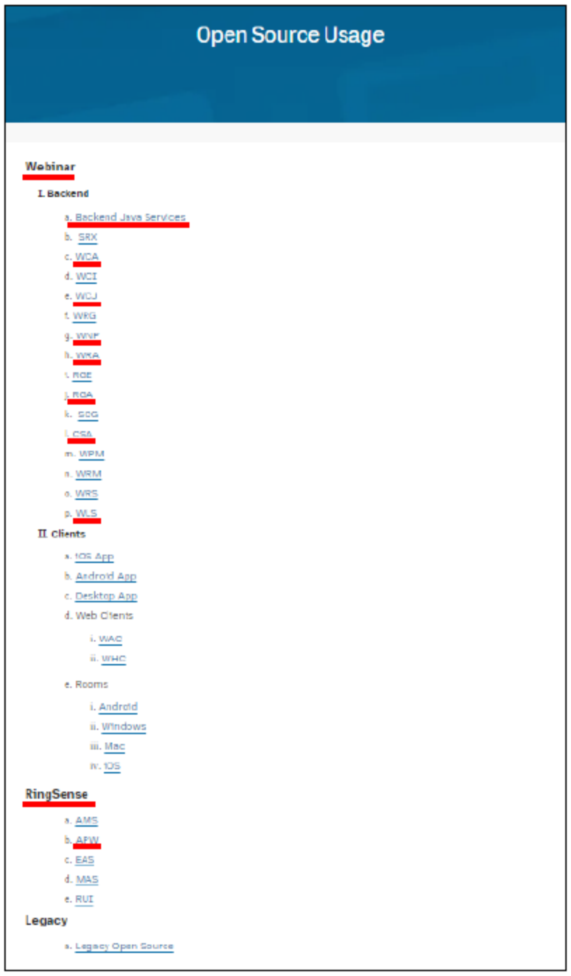



QOS Call Record Correlation
Real Time Call Detail Records
RTCPXR
Kafka
RealTime Indexed UCDR (Past 12 hours)
Correlated UCDR (12+ hours)
StreamSets
Spark Streaming
RealTime Indexed UCDR
HBase (Look Ups)
YARN
HDFS
cloudera
Correlated UCDR (12+ hours)
VERTICA
tableau
Indexed CDR with Common Key Loaded Directly into Vertica with Multiple CDRs
Correlated QOS Record with Meta Data for Call Generated after 12 Hours written to HDFS and Vertica
RingCentral





Source: https://youtu.be/KMs-1BgQB0Q?si=tIlpYhp4jqc2d_Xi (emphasis added).



Open Source Usage

Webinar

I. Backend

a. Backend Java Services
b. SRX
c. WCA
d. WCI
e. WCJ
f. WRG
g. WNF
h. WKA
i. RGE
j. RGA
k. SGG
l. CSA
m. WPM
n. WRM
o. WRS
p. WLS

II. Clients

a. iOS App
b. Android App
c. Desktop App
d. Web Clients
  i. WAC
  ii. WHC
e. Rooms
  i. Android
  ii. Windows
  iii. Mac
  iv. iOS

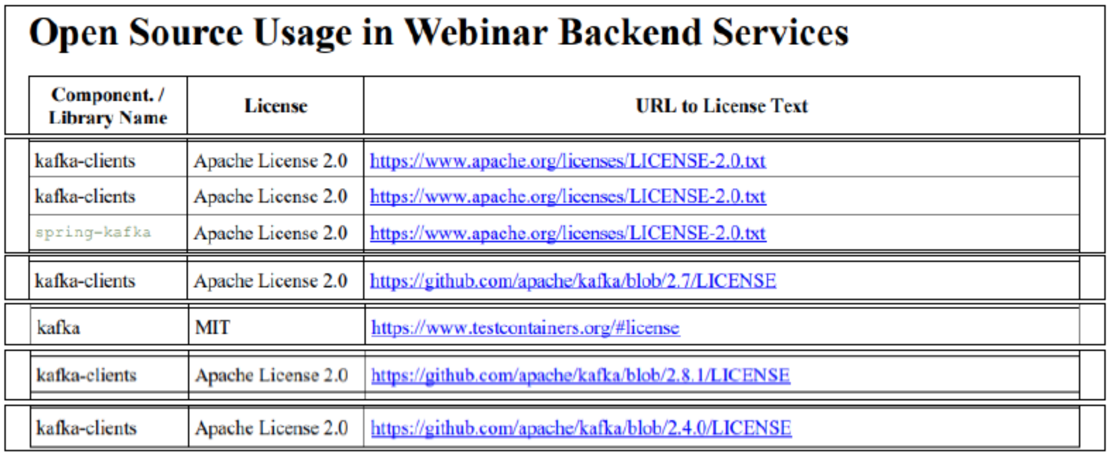
RingSense

a. AMS
b. APW
c. EAS
d. MAS
e. RUI

Legacy

a. Legacy Open Source

Source: https://www.ringcentral.com/legal/license_attribution.html (emphasis added).

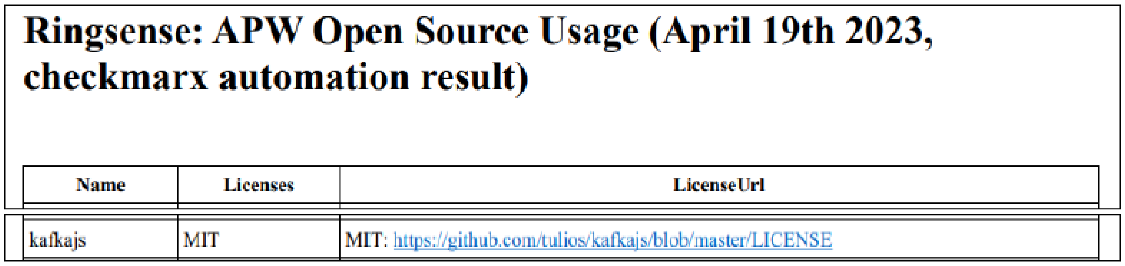


Open Source Usage in Webinar Backend Services

Component. / Library Name	License	URL to License Text
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
spring-kafka	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.7/LICENSE
kafka	MIT	https://www.testcontainers.org/#license
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.8.1/LICENSE
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.4.0/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/backend-java-services.pdf.[4]

[4] *See also* https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wca.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/



Ringsense: APW Open Source Usage (April 19th 2023, checkmarx automation result)

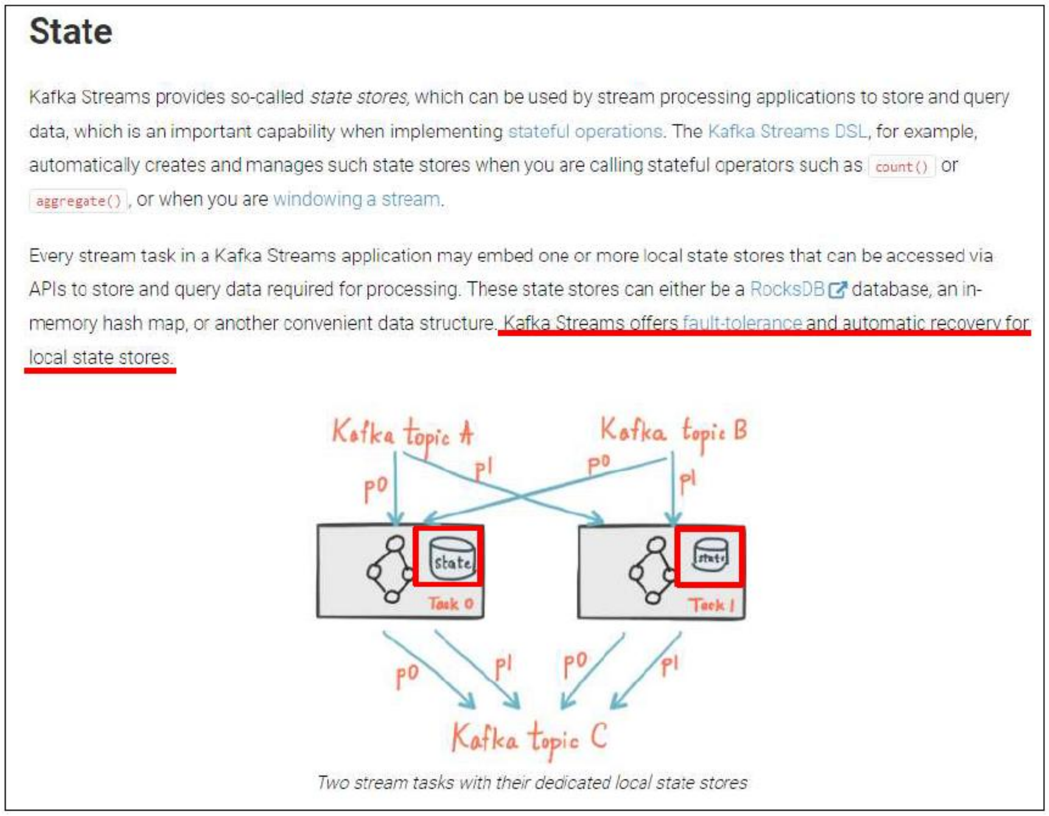
Name	Licenses	LicenseUrl
kafkajs	MIT	MIT: https://github.com/tulios/kafkajs/blob/master/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/apw.pdf.

69. Defendant directly infringes the ’945 Patent under 35 U.S.C. § 271(a) by using, making, offering for sale, selling, and/or importing the ’945 Accused Products, their components and processes, and/or products containing the same that incorporate the fundamental technologies covered by the ’945 Patent.

70. For example, Defendant infringes claim 1 of the ’945 Patent via the ’945 Accused Products. The ’945 Accused Products perform a “method for preserving state and reducing data loss.” For example, Kafka Streams performs a method for preserving state and reducing data loss.

wcj.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wnp.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wra.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/rga.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/csa.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wls-open-source-usage.pdf.



State

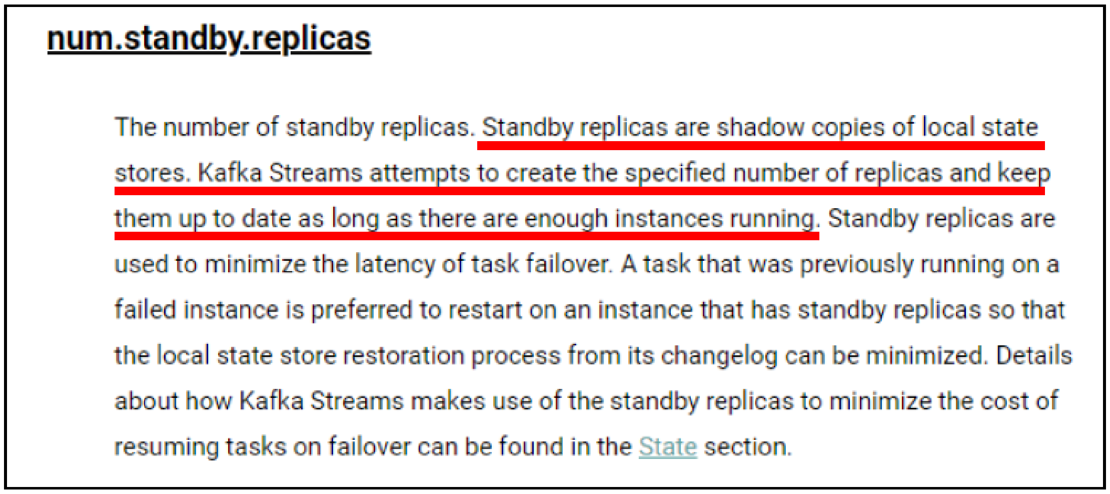
Kafka Streams provides so-called *state stores*, which can be used by stream processing applications to store and query data, which is an important capability when implementing stateful operations. The Kafka Streams DSL, for example, automatically creates and manages such state stores when you are calling stateful operators such as `count()` or `aggregate()`, or when you are windowing a stream.

Every stream task in a Kafka Streams application may embed one or more local state stores that can be accessed via APIs to store and query data required for processing. These state stores can either be a RocksDB database, an in-memory hash map, or another convenient data structure. Kafka Streams offers fault-tolerance and automatic recovery for local state stores.

Two stream tasks with their dedicated local state stores

Source: https://docs.confluent.io/platform/current/streams/architecture.html#state (emphasis added).

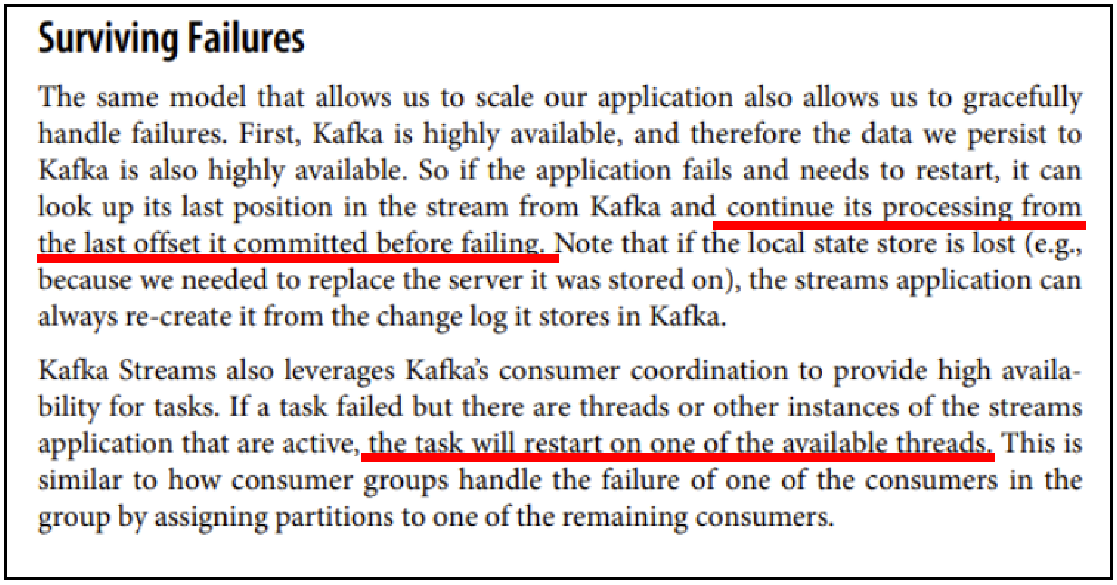
71. The ’945 Accused Products operate such that “upon detecting a commit in an active device, continuously copying all inbound data traffic before receipt at the active device to one or more buffers associated with a standby device until a next commit or failure, wherein the copied inbound data traffic has a destination address changed to that of the standby device.” For example, an active instance of a Kafka Streams Application commits state to a local state store that replicated to standby instances through a changelog topic. The data traffic inbound to the changelog is replicated (in alignment with Kafka’s topic replication principles) before the changelog topic is ever received at the active instance. When the state of the active instance is updated, the changelog topic (and its In-Sync Replicas) is updated. The replicated changelog data is addressed to the standby instances:



num.standby.replicas

The number of standby replicas. Standby replicas are shadow copies of local state stores. Kafka Streams attempts to create the specified number of replicas and keep them up to date as long as there are enough instances running. Standby replicas are used to minimize the latency of task failover. A task that was previously running on a failed instance is preferred to restart on an instance that has standby replicas so that the local state store restoration process from its changelog can be minimized. Details about how Kafka Streams makes use of the standby replicas to minimize the cost of resuming tasks on failover can be found in the State section.

Source: https://kafka.apache.org/10/documentation/streams/developer-guide/config-streams.html#num-standby-replicas (emphasis added).

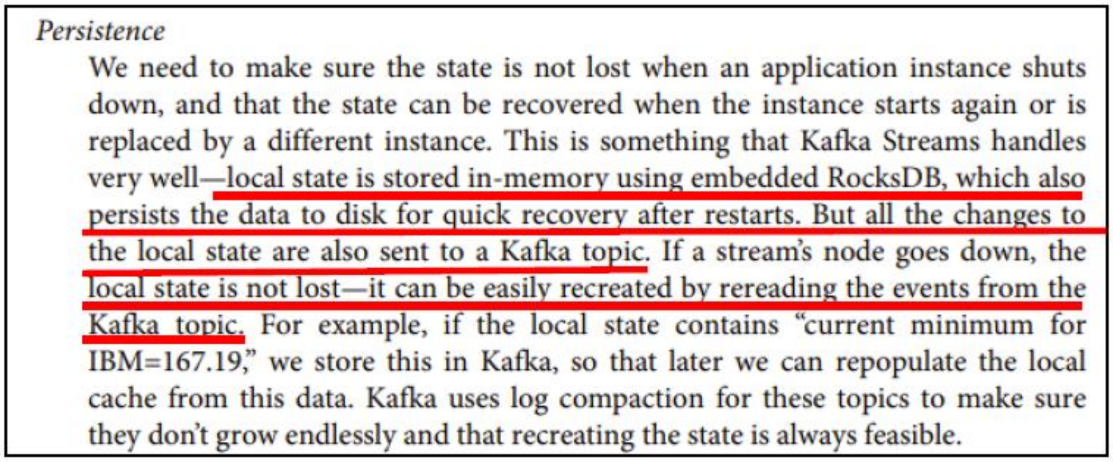


Surviving Failures

The same model that allows us to scale our application also allows us to gracefully handle failures. First, Kafka is highly available, and therefore the data we persist to Kafka is also highly available. So if the application fails and needs to restart, it can look up its last position in the stream from Kafka and continue its processing from the last offset it committed before failing. Note that if the local state store is lost (e.g., because we needed to replace the server it was stored on), the streams application can always re-create it from the change log it stores in Kafka.

Kafka Streams also leverages Kafka's consumer coordination to provide high availability for tasks. If a task failed but there are threads or other instances of the streams application that are active, the task will restart on one of the available threads. This is similar to how consumer groups handle the failure of one of the consumers in the group by assigning partitions to one of the remaining consumers.

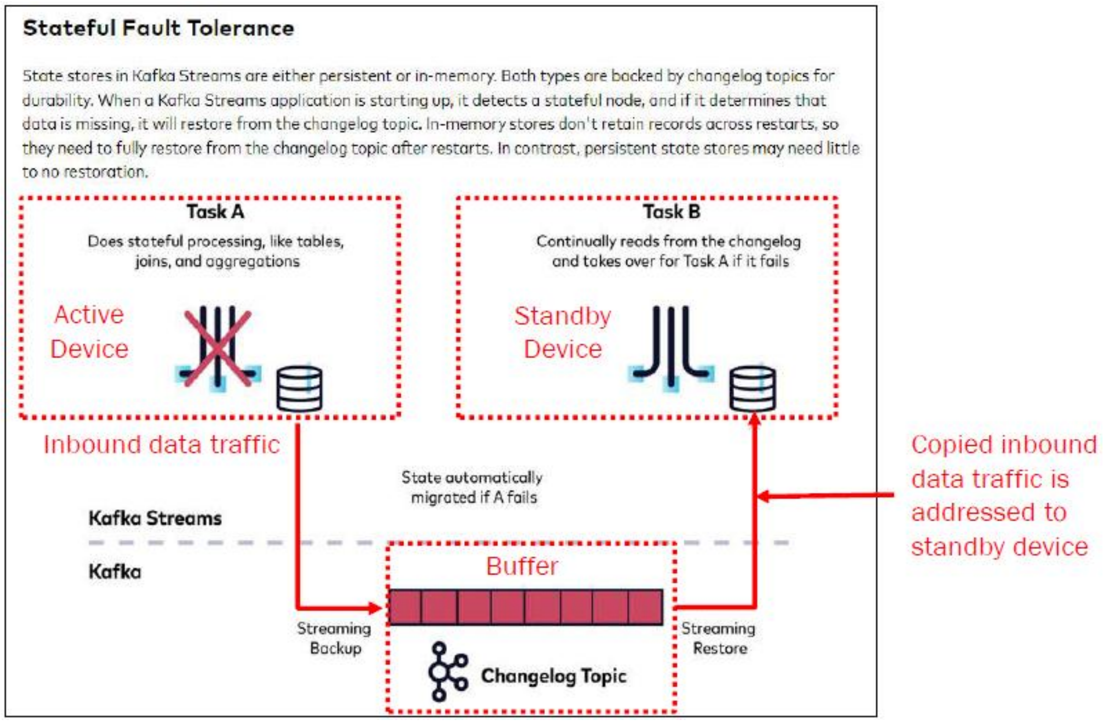
Source: https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf (emphasis added).



Persistence

We need to make sure the state is not lost when an application instance shuts down, and that the state can be recovered when the instance starts again or is replaced by a different instance. This is something that Kafka Streams handles very well—local state is stored in-memory using embedded RocksDB, which also persists the data to disk for quick recovery after restarts. But all the changes to the local state are also sent to a Kafka topic. If a stream's node goes down, the local state is not lost—it can be easily recreated by rereading the events from the Kafka topic. For example, if the local state contains "current minimum for IBM=167.19," we store this in Kafka, so that later we can repopulate the local cache from this data. Kafka uses log compaction for these topics to make sure they don't grow endlessly and that recreating the state is always feasible.

Source: https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf (emphasis added).

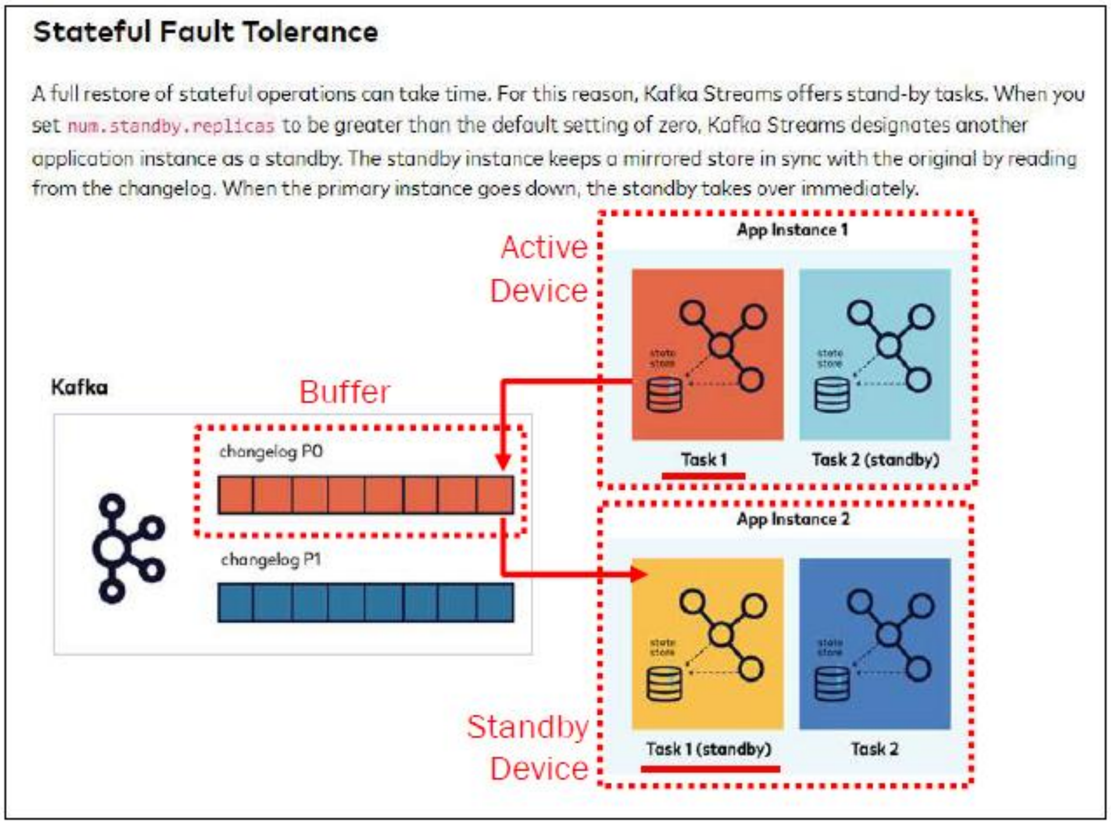


Stateful Fault Tolerance

State stores in Kafka Streams are either persistent or in-memory. Both types are backed by changelog topics for durability. When a Kafka Streams application is starting up, it detects a stateful node, and if it determines that data is missing, it will restore from the changelog topic. In-memory stores don't retain records across restarts, so they need to fully restore from the changelog topic after restarts. In contrast, persistent state stores may need little to no restoration.

Task A
Does stateful processing, like tables, joins, and aggregations
Active Device
Task B
Continually reads from the changelog and takes over for Task A if it fails
Standby Device
Inbound data traffic
State automatically migrated if A fails
Copied inbound data traffic is addressed to standby device
Kafka Streams
Kafka
Buffer
Streaming Backup
Streaming Restore
Changelog Topic

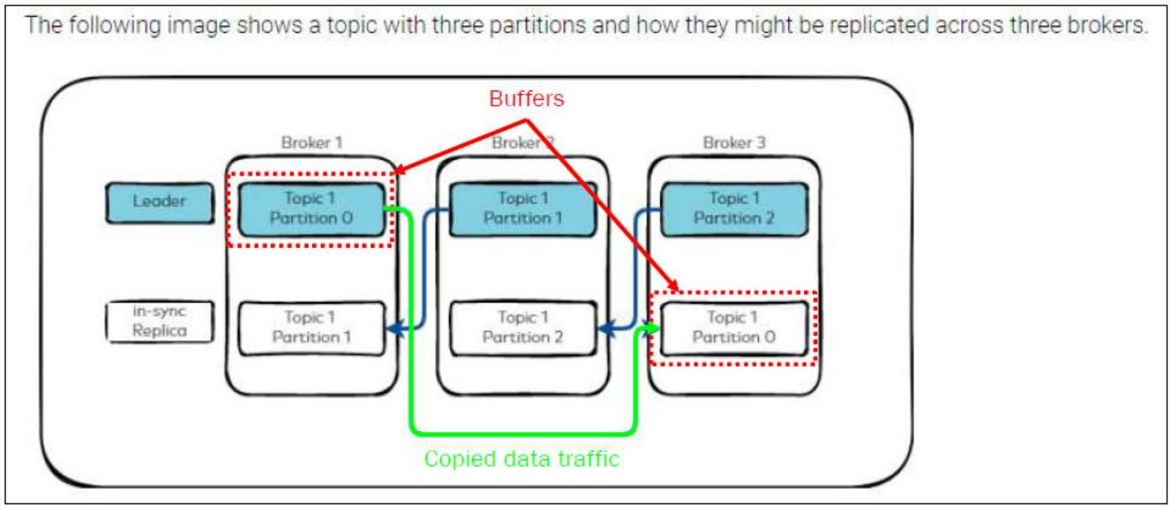
Source: https://developer.confluent.io/courses/kafka-streams/stateful-fault-tolerance/ (emphasis added).



Stateful Fault Tolerance

A full restore of stateful operations can take time. For this reason, Kafka Streams offers stand-by tasks. When you set `num.standby.replicas` to be greater than the default setting of zero, Kafka Streams designates another application instance as a standby. The standby instance keeps a mirrored store in sync with the original by reading from the changelog. When the primary instance goes down, the standby takes over immediately.

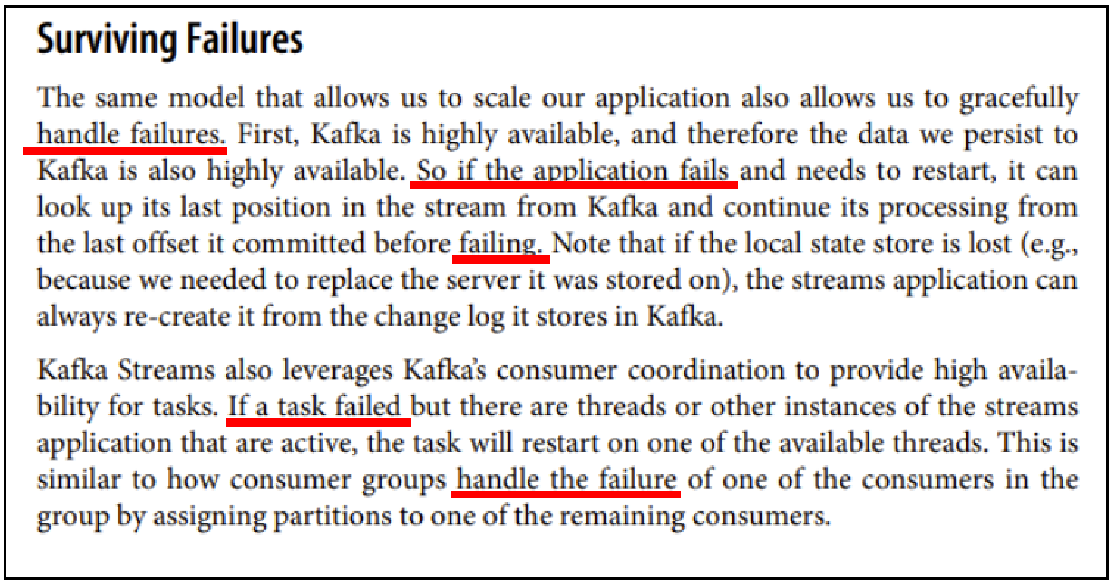
Source: https://developer.confluent.io/courses/kafka-streams/stateful-fault-tolerance/ (emphasis added).



The following image shows a topic with three partitions and how they might be replicated across three brokers.

Source: https://docs.confluent.io/kafka/design/replication.html (emphasis added).

72. The ’945 Accused Products detect “a failure.” For example, Kafka Streams is fault-tolerant, so it detects failures:

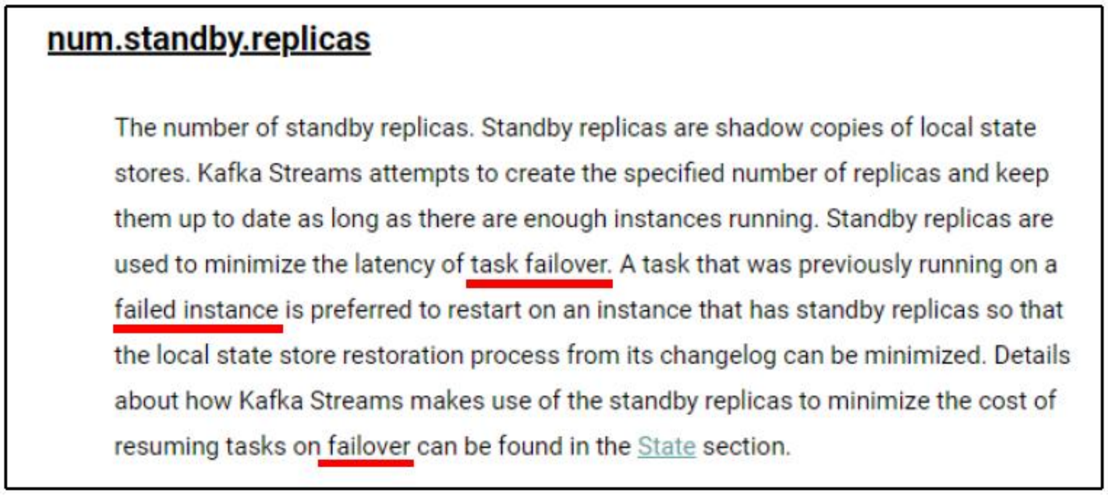


Surviving Failures

The same model that allows us to scale our application also allows us to gracefully handle failures. First, Kafka is highly available, and therefore the data we persist to Kafka is also highly available. So if the application fails and needs to restart, it can look up its last position in the stream from Kafka and continue its processing from the last offset it committed before failing. Note that if the local state store is lost (e.g., because we needed to replace the server it was stored on), the streams application can always re-create it from the change log it stores in Kafka.

Kafka Streams also leverages Kafka's consumer coordination to provide high availability for tasks. If a task failed but there are threads or other instances of the streams application that are active, the task will restart on one of the available threads. This is similar to how consumer groups handle the failure of one of the consumers in the group by assigning partitions to one of the remaining consumers.

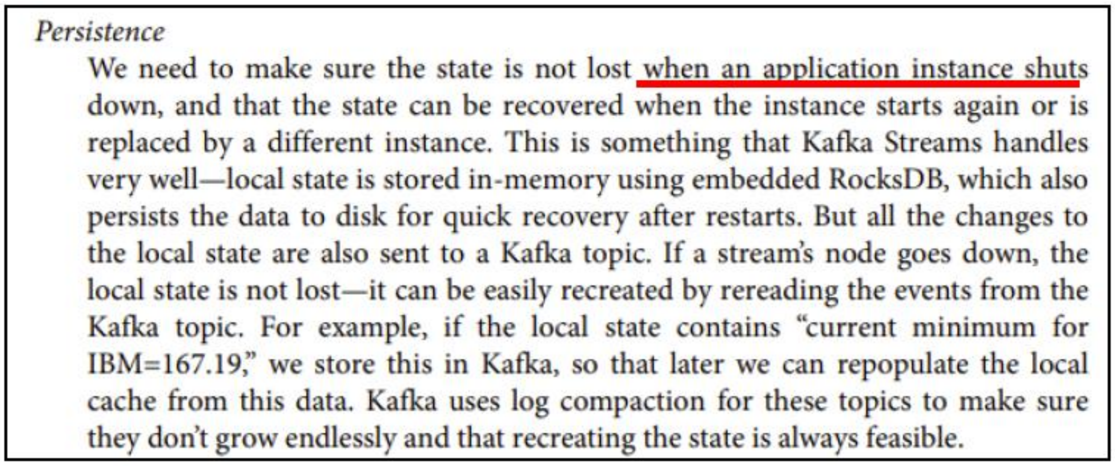
Source: https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf (emphasis added).



num.standby.replicas

The number of standby replicas. Standby replicas are shadow copies of local state stores. Kafka Streams attempts to create the specified number of replicas and keep them up to date as long as there are enough instances running. Standby replicas are used to minimize the latency of task failover. A task that was previously running on a failed instance is preferred to restart on an instance that has standby replicas so that the local state store restoration process from its changelog can be minimized. Details about how Kafka Streams makes use of the standby replicas to minimize the cost of resuming tasks on failover can be found in the State section.

Source: https://kafka.apache.org/10/documentation/streams/developer-guide/config-streams.html#num-standby-replicas (emphasis added).



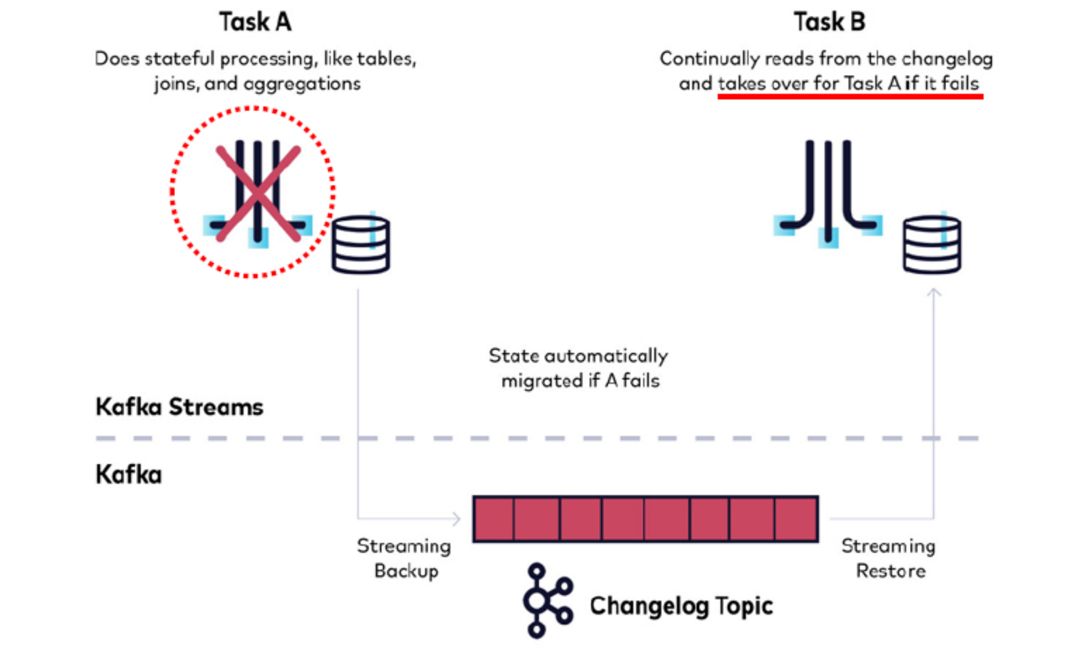
Persistence

We need to make sure the state is not lost when an application instance shuts down, and that the state can be recovered when the instance starts again or is replaced by a different instance. This is something that Kafka Streams handles very well—local state is stored in-memory using embedded RocksDB, which also persists the data to disk for quick recovery after restarts. But all the changes to the local state are also sent to a Kafka topic. If a stream's node goes down, the local state is not lost—it can be easily recreated by rereading the events from the Kafka topic. For example, if the local state contains "current minimum for IBM=167.19," we store this in Kafka, so that later we can repopulate the local cache from this data. Kafka uses log compaction for these topics to make sure they don't grow endlessly and that recreating the state is always feasible.

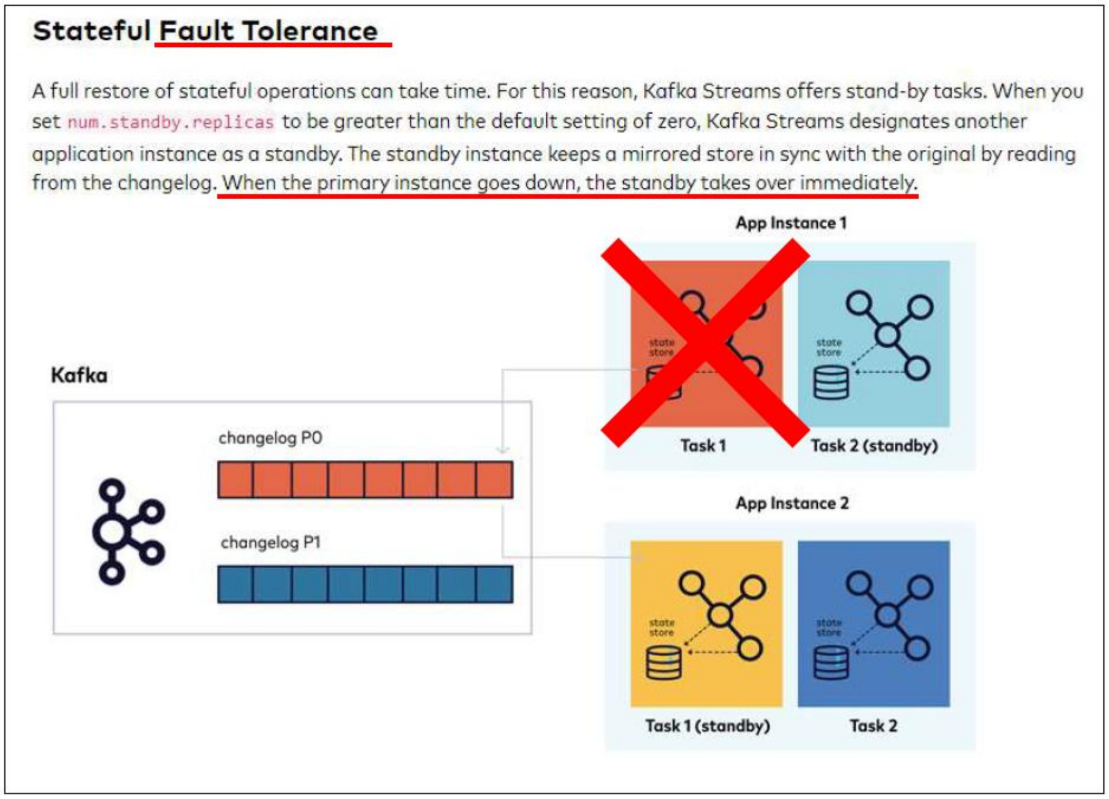
Source: https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf (emphasis added).

Stateful Fault Tolerance

State stores in Kafka Streams are either persistent or in-memory. Both types are backed by changelog topics for durability. When a Kafka Streams application is starting up, it detects a stateful node, and if it determines that data is missing, it will restore from the changelog topic. In-memory stores don't retain records across restarts, so they need to fully restore from the changelog topic after restarts. In contrast, persistent state stores may need little to no restoration.


Task A
Does stateful processing, like tables, joins, and aggregations
Task B
Continually reads from the changelog and takes over for Task A if it fails
State automatically migrated if A fails
Kafka Streams
Kafka
Streaming Backup
Streaming Restore
Changelog Topic

Source: https://developer.confluent.io/courses/kafka-streams/stateful-fault-tolerance/ (emphasis added).

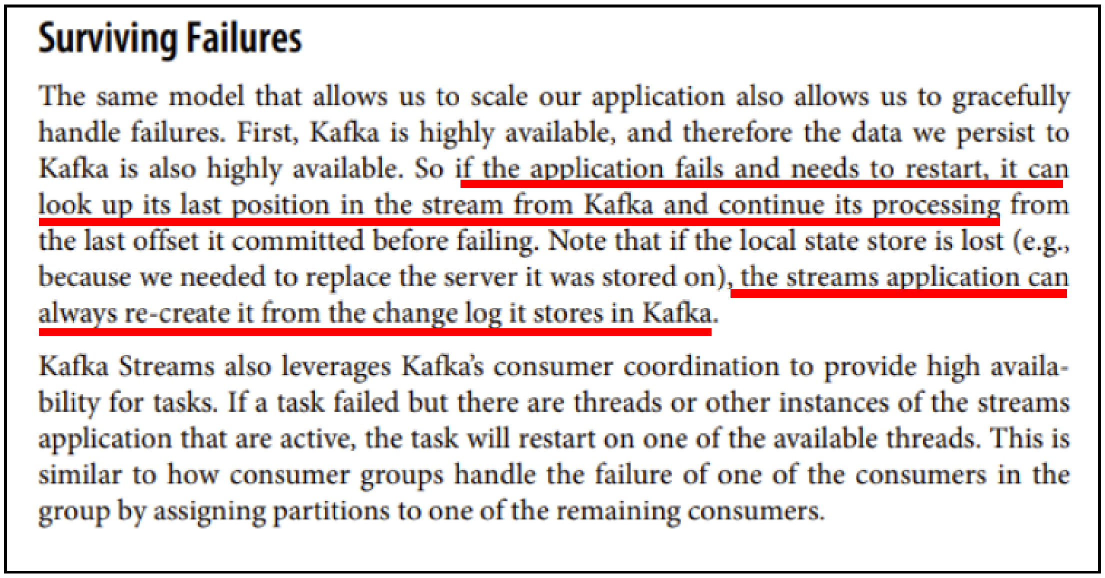


Stateful Fault Tolerance

A full restore of stateful operations can take time. For this reason, Kafka Streams offers stand-by tasks. When you set `num.standby.replicas` to be greater than the default setting of zero, Kafka Streams designates another application instance as a standby. The standby instance keeps a mirrored store in sync with the original by reading from the changelog. When the primary instance goes down, the standby takes over immediately.

Source: https://developer.confluent.io/courses/kafka-streams/stateful-fault-tolerance/ (emphasis added).

73. The '945 Accused Products thereafter replay "copied data traffic to restore the standby device to a current state of a failed device." For example, Kafka replays the changelog to restore standby application instances to the current state of a failed task:

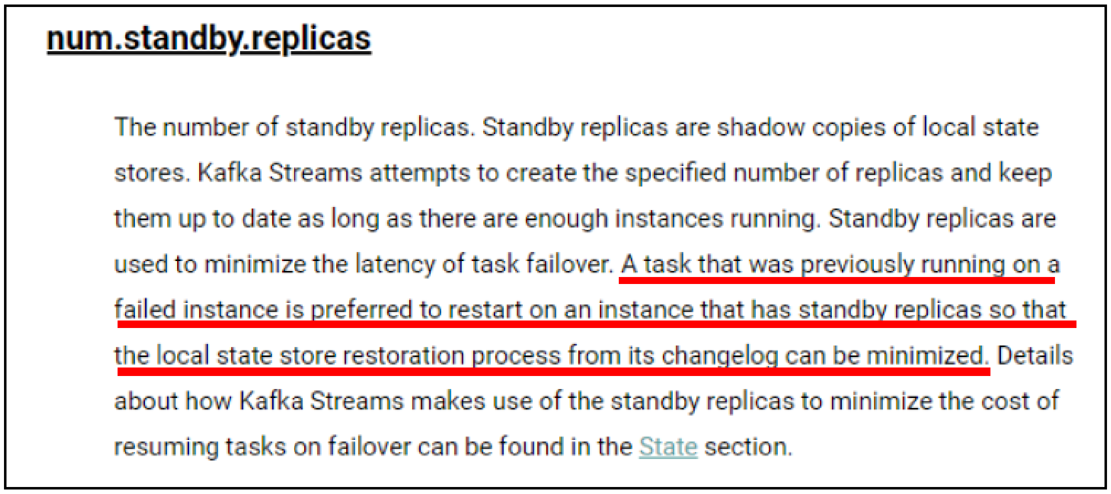


Surviving Failures

The same model that allows us to scale our application also allows us to gracefully handle failures. First, Kafka is highly available, and therefore the data we persist to Kafka is also highly available. So if the application fails and needs to restart, it can look up its last position in the stream from Kafka and continue its processing from the last offset it committed before failing. Note that if the local state store is lost (e.g., because we needed to replace the server it was stored on), the streams application can always re-create it from the change log it stores in Kafka.

Kafka Streams also leverages Kafka's consumer coordination to provide high availability for tasks. If a task failed but there are threads or other instances of the streams application that are active, the task will restart on one of the available threads. This is similar to how consumer groups handle the failure of one of the consumers in the group by assigning partitions to one of the remaining consumers.

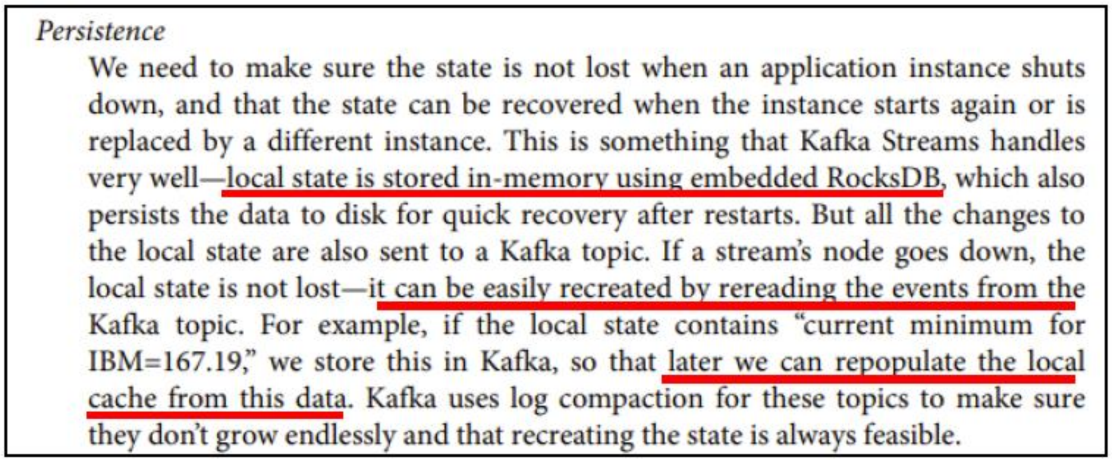
Source: https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf (emphasis added).



num.standby.replicas

The number of standby replicas. Standby replicas are shadow copies of local state stores. Kafka Streams attempts to create the specified number of replicas and keep them up to date as long as there are enough instances running. Standby replicas are used to minimize the latency of task failover. A task that was previously running on a failed instance is preferred to restart on an instance that has standby replicas so that the local state store restoration process from its changelog can be minimized. Details about how Kafka Streams makes use of the standby replicas to minimize the cost of resuming tasks on failover can be found in the State section.

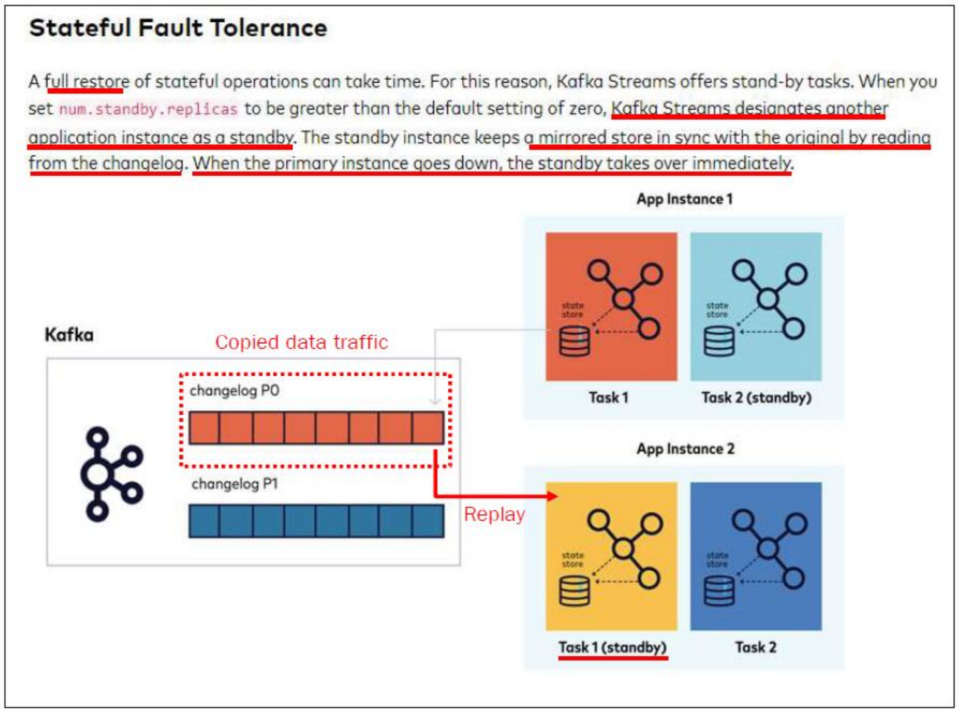
Source: https://kafka.apache.org/10/documentation/streams/developer-guide/config-streams.html#num-standby-replicas (emphasis added).



Persistence

We need to make sure the state is not lost when an application instance shuts down, and that the state can be recovered when the instance starts again or is replaced by a different instance. This is something that Kafka Streams handles very well—local state is stored in-memory using embedded RocksDB, which also persists the data to disk for quick recovery after restarts. But all the changes to the local state are also sent to a Kafka topic. If a stream's node goes down, the local state is not lost—it can be easily recreated by rereading the events from the Kafka topic. For example, if the local state contains "current minimum for IBM=167.19," we store this in Kafka, so that later we can repopulate the local cache from this data. Kafka uses log compaction for these topics to make sure they don't grow endlessly and that recreating the state is always feasible.

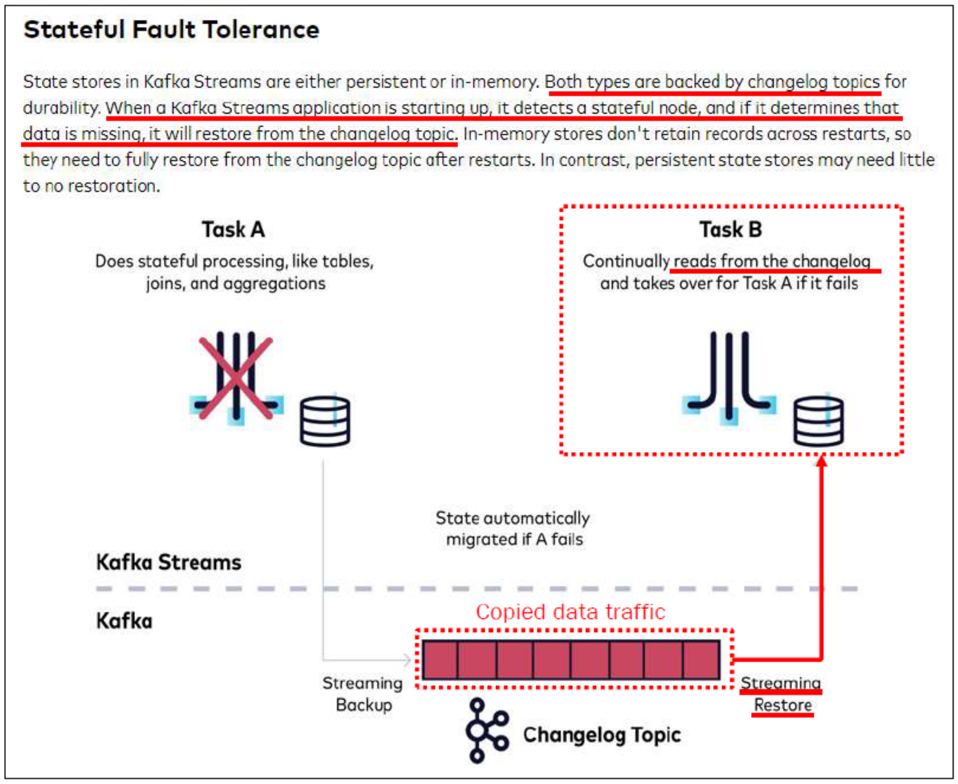
Source: https://www.confluent.io/wp-content/uploads/confluent-kafka-definitive-guide-complete.pdf (emphasis added).



Stateful Fault Tolerance

A full restore of stateful operations can take time. For this reason, Kafka Streams offers stand-by tasks. When you set `num.standby.replicas` to be greater than the default setting of zero, Kafka Streams designates another application instance as a standby. The standby instance keeps a mirrored store in sync with the original by reading from the changelog. When the primary instance goes down, the standby takes over immediately.

Source: https://developer.confluent.io/courses/kafka-streams/stateful-fault-tolerance/ (emphasis added).



Stateful Fault Tolerance

State stores in Kafka Streams are either persistent or in-memory. Both types are backed by changelog topics for durability. When a Kafka Streams application is starting up, it detects a stateful node, and if it determines that data is missing, it will restore from the changelog topic. In-memory stores don't retain records across restarts, so they need to fully restore from the changelog topic after restarts. In contrast, persistent state stores may need little to no restoration.

Task A
Does stateful processing, like tables, joins, and aggregations
Task B
Continually reads from the changelog and takes over for Task A if it fails
State automatically migrated if A fails
Kafka Streams
Kafka
Copied data traffic
Streaming Backup
Streaming Restore
Changelog Topic

Source: https://developer.confluent.io/courses/kafka-streams/stateful-fault-tolerance/ (emphasis added).

74. The technology discussion above and the exemplary ’945 Accused Products provide context for Plaintiff’s infringement allegations.

75. At a minimum, Defendant has known of the ’945 Patent at least as early as the filing date of the Complaint. In addition, Defendant has known about the ’945 Patent since at least March 8, 2024, when Plaintiff sent correspondence to Defendant via FedEx alerting Defendant to its infringement.

76. On information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has actively induced, under U.S.C. § 271(b), its distributors, customers, subsidiaries, importers, and/or consumers that import, purchase, or sell the ’945 Accused Products that include or are made using all of the limitations of one or more claims of the ’945 Patent to directly infringe one or more claims of the ’945 Patent (e.g., claim 1, as discussed above) by using, offering for sale, selling, and/or importing the ’945 Accused Products. Since at least the notice provided on the above-mentioned date, Defendant does so with knowledge, or with willful blindness of the fact, that the induced acts constitute infringement of the ’945 Patent. Defendant intends to cause, and has taken affirmative steps to induce infringement by its distributors, importers, customers, subsidiaries, and/or consumers by at least, *inter alia*, creating advertisements that promote the infringing use of the ’945 Accused Products, creating and/or maintaining established distribution channels for the ’945 Accused Products into and within the United States, manufacturing the ’945 Accused Products in conformity with U.S. laws and regulations, distributing or making available instructions or manuals for these products to purchasers and prospective buyers, and testing the ’945 Accused Products, and/or providing technical support, replacement parts, or services for these products to these purchasers in the United States. For example, Defendant configures Apache Kafka in the ’945 Accused Products to

contain specific instructions, such as source code and configuration files, that cause such products to support at least its Webinar and RingSense products.

77. In the alternative, on information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has contributorily infringed, under U.S.C. § 271(c), one or more claims of the ’945 Patent. For example, Defendant contributes to the direct infringement of such claims by distributors, customers, subsidiaries, importers, and/or consumers that use, import, purchase, or sell the ’945 Accused Products. To the extent that the ’945 Accused Products do not directly infringe one or more claims of the ’945 Patent, such products contain instructions, such as source code, that are especially adapted to cause the ’945 Accused Products to operate in an infringing manner. Such instructions are specifically designed to cause the ’945 Accused Products to provide and utilize Apache Kafka in an infringing manner and are a material part of the invention of the ’945 Patent and are not a staple article of commerce suitable for substantial non-infringing use.

78. On information and belief, despite having knowledge of the ’945 Patent and knowledge that it is directly and/or indirectly infringing one or more claims of the ’945 Patent, Defendant has nevertheless continued its infringing conduct and disregarded an objectively high likelihood of infringement. Defendant’s infringing activities relative to the ’945 Patent have been, and continue to be, willful, wanton, malicious, in bad-faith, deliberate, consciously wrongful, flagrant, and an egregious case of misconduct beyond typical infringement such that Plaintiff is entitled under 35 U.S.C. § 284 to enhanced damages up to three times the amount found or assessed.

79. Plaintiff has been damaged as a result of Defendant’s infringing conduct described in this Count. Defendant is, thus, liable to Plaintiff in an amount that adequately compensates

Plaintiff for Defendant's infringements, which, by law, cannot be less than a reasonable royalty, together with interest and costs as fixed by this Court under 35 U.S.C. § 284.

COUNT IV
(Infringement of U.S. Patent No. 9,026,836)

80. Plaintiff incorporates the preceding paragraphs herein by reference.

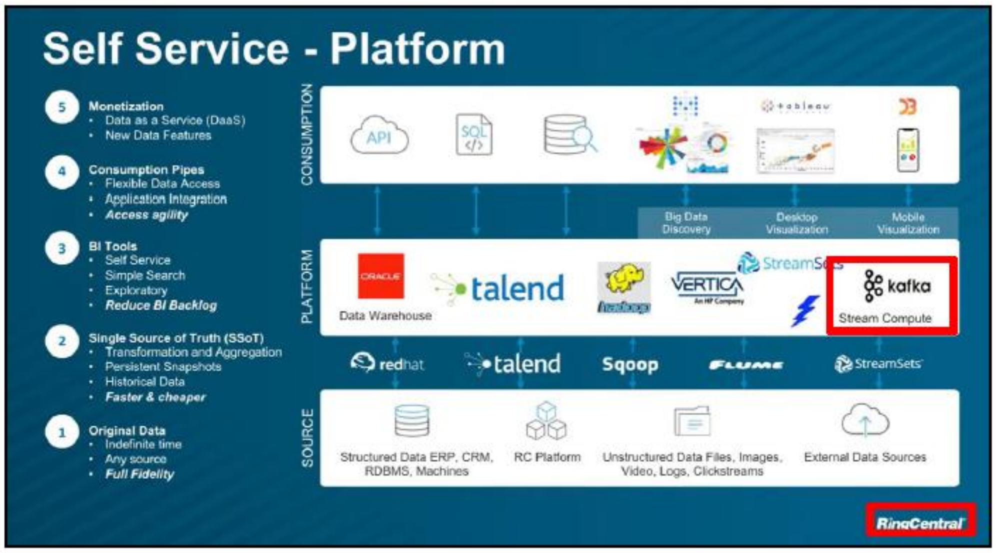
81. This cause of action arises under the patent laws of the United States, and, in particular, 35 U.S.C. §§ 271, *et seq*.

82. Plaintiff is the assignee of the '836 Patent, with ownership of all substantial rights in the '836 Patent, including the right to exclude others and to enforce, sue, and recover damages for past, present, and future infringements.

83. The '836 Patent is valid, enforceable, and was duly issued in full compliance with Title 35 of the United States Code after a full and fair examination.

84. Defendant has and continues to directly and/or indirectly infringe (by inducing infringement and/or contributing to infringement) one or more claims of the '836 Patent in this judicial district and elsewhere in Delaware and the United States.

85. Defendant designs, offers for sale, uses, and sells services, such as Apache Kafka ("the '836 Accused Products"), in a manner that infringes the '836 Patent. For example, Defendant uses Apache Kafka to support at least its Webinar and RingSense products:



Self Service - Platform

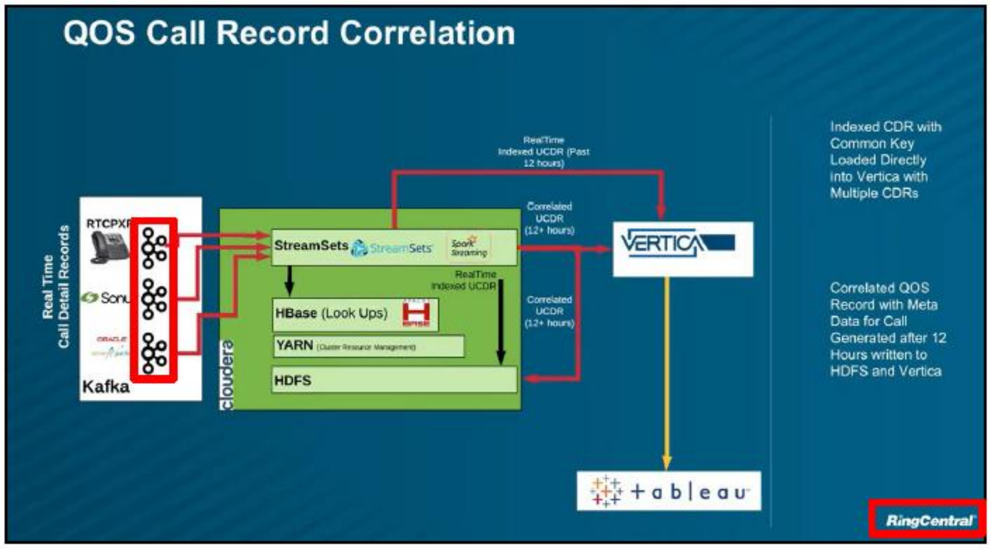
5 **Monetization**
- Data as a Service (DaaS)
- New Data Features

4 **Consumption Pipes**
- Flexible Data Access
- Application Integration
- ***Access agility***

3 **BI Tools**
- Self Service
- Simple Search
- Exploratory
- ***Reduce BI Backlog***

2 **Single Source of Truth (SSoT)**
- Transformation and Aggregation
- Persistent Snapshots
- Historical Data
- ***Faster & cheaper***

1 **Original Data**
- Indefinite time
- Any source
- ***Full Fidelity***

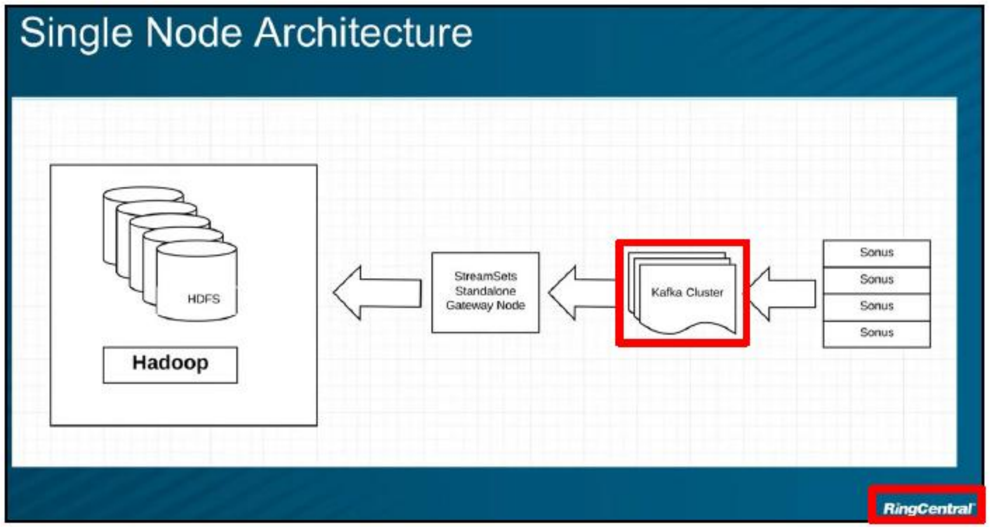


QOS Call Record Correlation

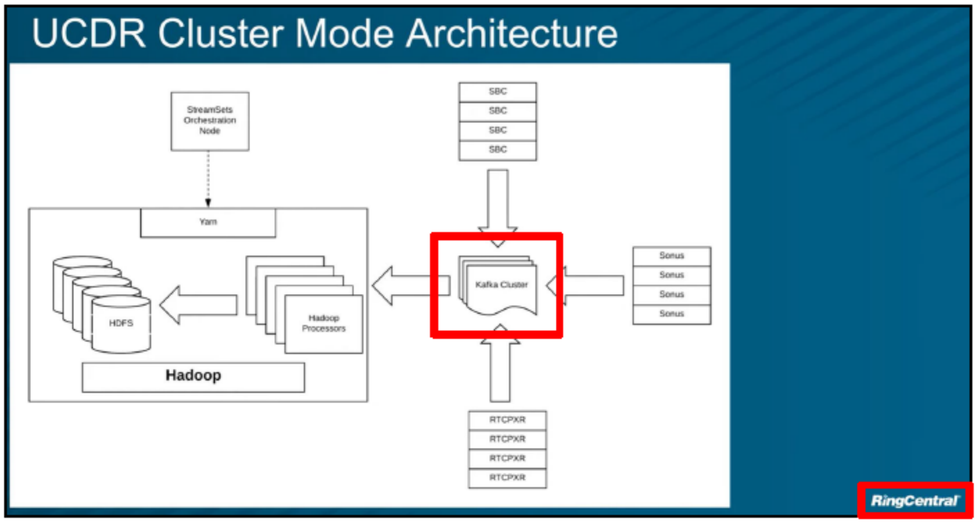
Real Time Call Detail Records
RTCPXR
Sonus
Kafka
cloudera
StreamSets
Spark Streaming
HBase (Look Ups)
YARN (Cluster Resource Management)
HDFS
RealTime Indexed UCDR
RealTime Indexed UCDR (Past 12 hours)
Correlated UCDR (12+ hours)
Correlated UCDR (12+ hours)
VERTICA
tableau
RingCentral

Indexed CDR with Common Key Loaded Directly into Vertica with Multiple CDRs

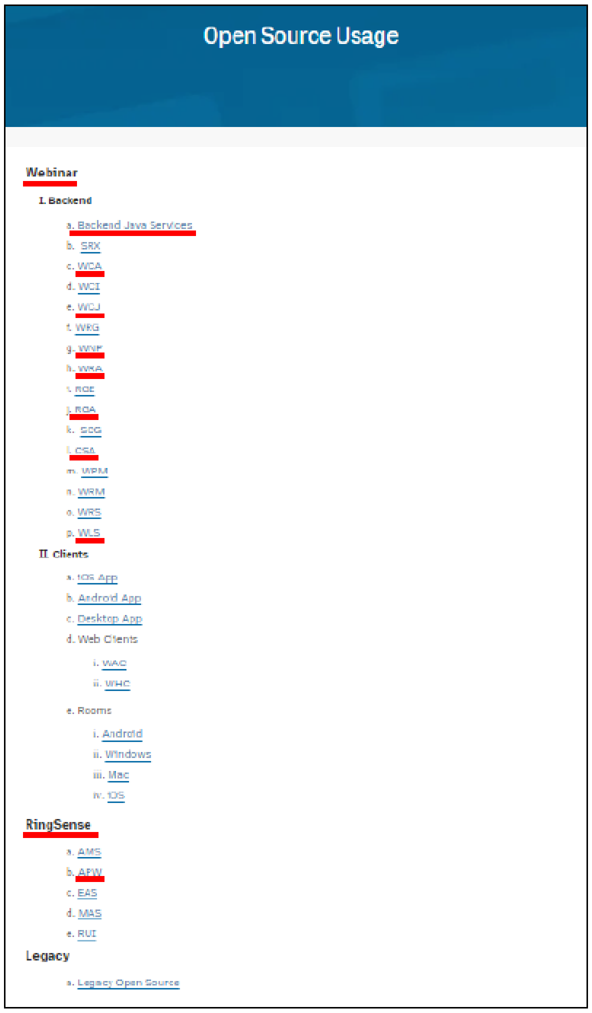
Correlated QOS Record with Meta Data for Call Generated after 12 Hours written to HDFS and Vertica



Single Node Architecture



Source: https://youtu.be/KMs-1BgQB0Q?si=tIlpYhp4jqc2d_Xi (emphasis added).



Open Source Usage

Webinar

I. Backend

a. Backend Java Services
b. SRX
c. WCA
d. WCI
e. WCJ
f. WRG
g. WNF
h. WRA
i. RGE
j. RGA
k. SDG
l. CSA
m. WPM
n. WRM
o. WRS
p. WLS

II. Clients

a. iOS App
b. Android App
c. Desktop App
d. Web Clients
   i. WAC
   ii. WHC
e. Rooms
   i. Android
   ii. Windows
   iii. Mac
   iv. iOS

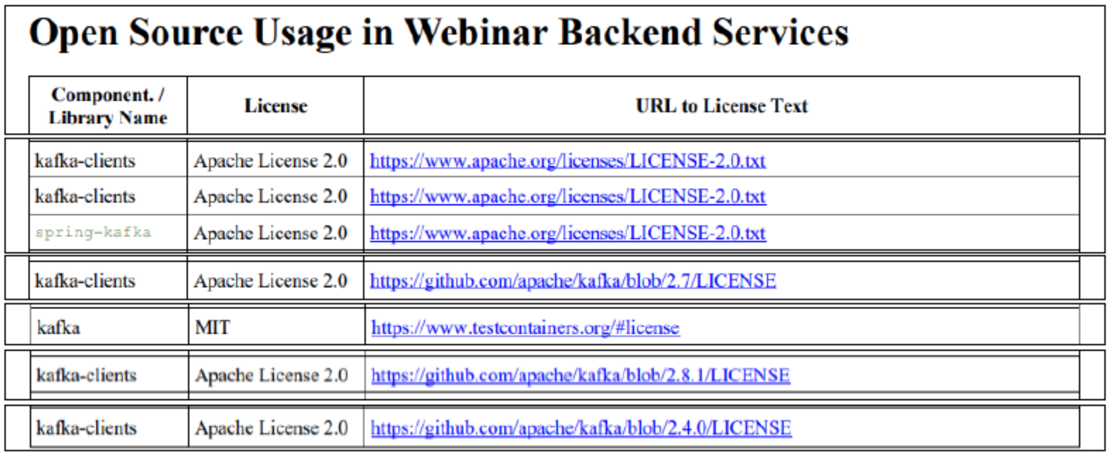
RingSense

a. AMS
b. APW
c. EAS
d. MAS
e. RUI

Legacy

a. Legacy Open Source

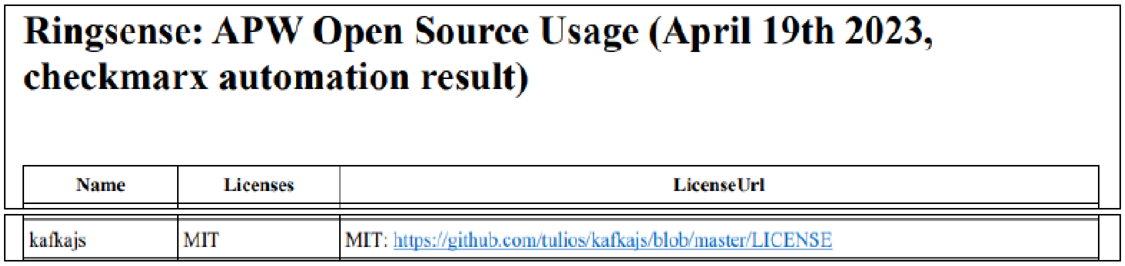
Source: https://www.ringcentral.com/legal/license_attribution.html (emphasis added).



Open Source Usage in Webinar Backend Services

Component. / Library Name	License	URL to License Text
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
spring-kafka	Apache License 2.0	https://www.apache.org/licenses/LICENSE-2.0.txt
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.7/LICENSE
kafka	MIT	https://www.testcontainers.org/#license
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.8.1/LICENSE
kafka-clients	Apache License 2.0	https://github.com/apache/kafka/blob/2.4.0/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/backend-java-services.pdf.[5]



Ringsense: APW Open Source Usage (April 19th 2023, checkmarx automation result)

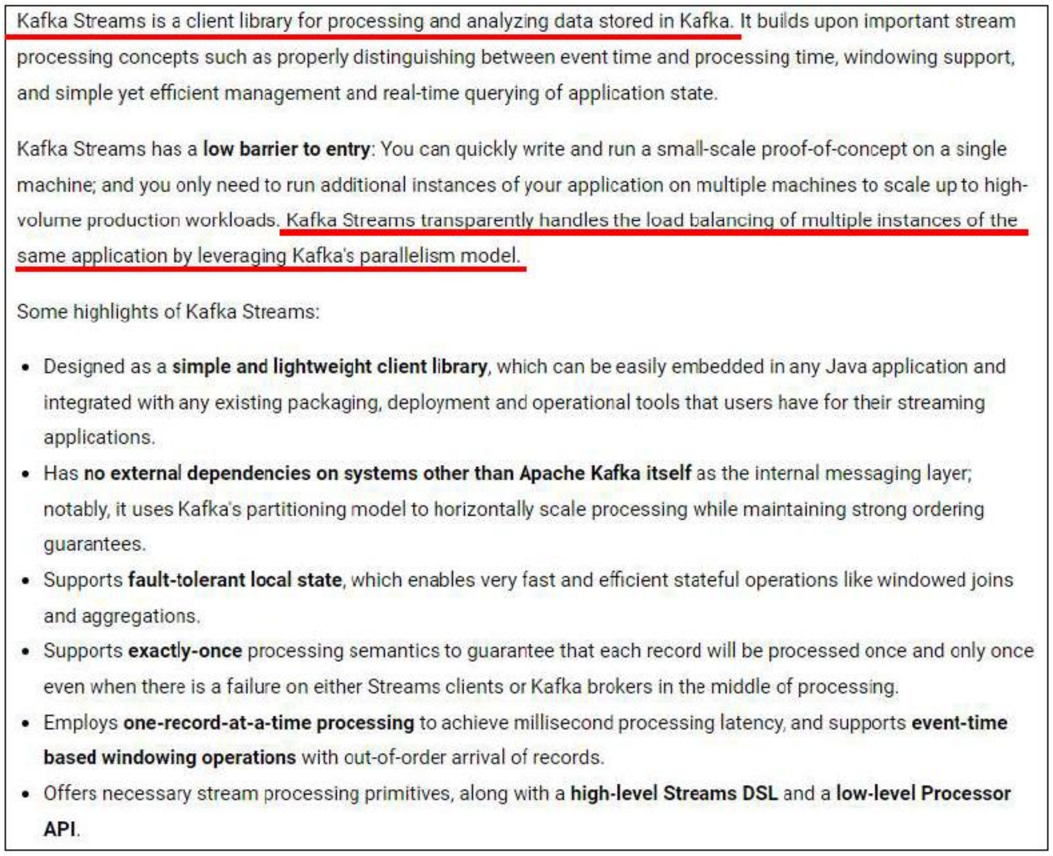
Name	Licenses	LicenseUrl
kafkajs	MIT	MIT: https://github.com/tulios/kafkajs/blob/master/LICENSE

Source: https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/apw.pdf.

86. Defendant directly infringes the ’836 Patent under 35 U.S.C. § 271(a) by using, making, offering for sale, selling, and/or importing the ’836 Accused Products, their components and processes, and/or products containing the same that incorporate the fundamental technologies covered by the ’836 Patent.

[5] *See also* https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wca.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wcj.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wnp.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wra.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/rga.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/csa.pdf; https://assets.ringcentral.com/legal/intellectual-property/open-source-attribution/wls-open-source-usage.pdf.

87. For example, Defendant infringes claim 1 of the ’836 Patent via the ’836 Accused Products. The ’836 Accused Products perform a method of restoration in response to application failure.

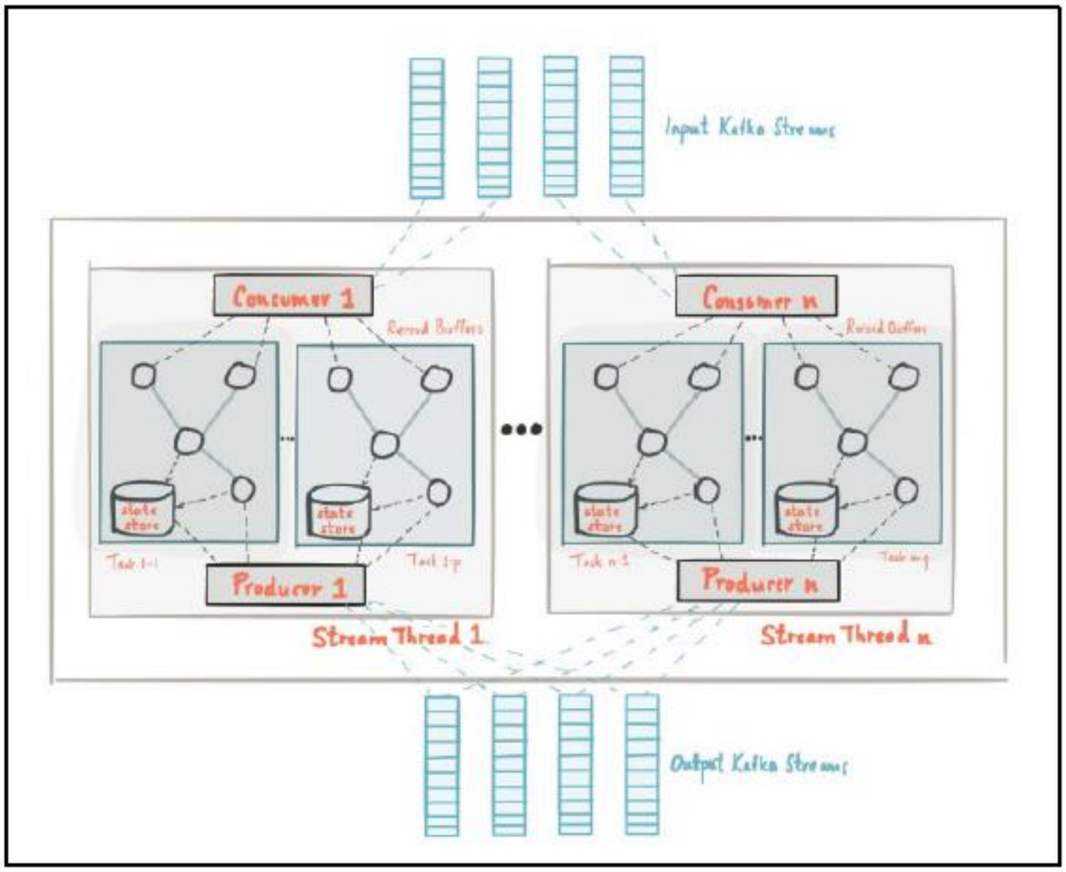

Kafka Streams is a client library for processing and analyzing data stored in Kafka. It builds upon important stream processing concepts such as properly distinguishing between event time and processing time, windowing support, and simple yet efficient management and real-time querying of application state.

Kafka Streams has a **low barrier to entry**: You can quickly write and run a small-scale proof-of-concept on a single machine; and you only need to run additional instances of your application on multiple machines to scale up to high-volume production workloads. Kafka Streams transparently handles the load balancing of multiple instances of the same application by leveraging Kafka's parallelism model.

Some highlights of Kafka Streams:

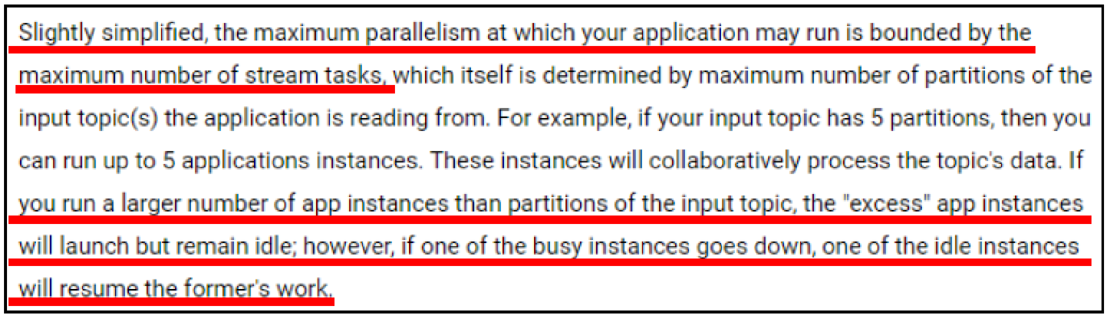
- Designed as a **simple and lightweight client library**, which can be easily embedded in any Java application and integrated with any existing packaging, deployment and operational tools that users have for their streaming applications.
- Has **no external dependencies on systems other than Apache Kafka itself** as the internal messaging layer; notably, it uses Kafka's partitioning model to horizontally scale processing while maintaining strong ordering guarantees.
- Supports **fault-tolerant local state**, which enables very fast and efficient stateful operations like windowed joins and aggregations.
- Supports **exactly-once** processing semantics to guarantee that each record will be processed once and only once even when there is a failure on either Streams clients or Kafka brokers in the middle of processing.
- Employs **one-record-at-a-time processing** to achieve millisecond processing latency, and supports **event-time based windowing operations** with out-of-order arrival of records.
- Offers necessary stream processing primitives, along with a **high-level Streams DSL** and a **low-level Processor API**.

Source: https://kafka.apache.org/36/documentation/streams/core-concepts (emphasis added).

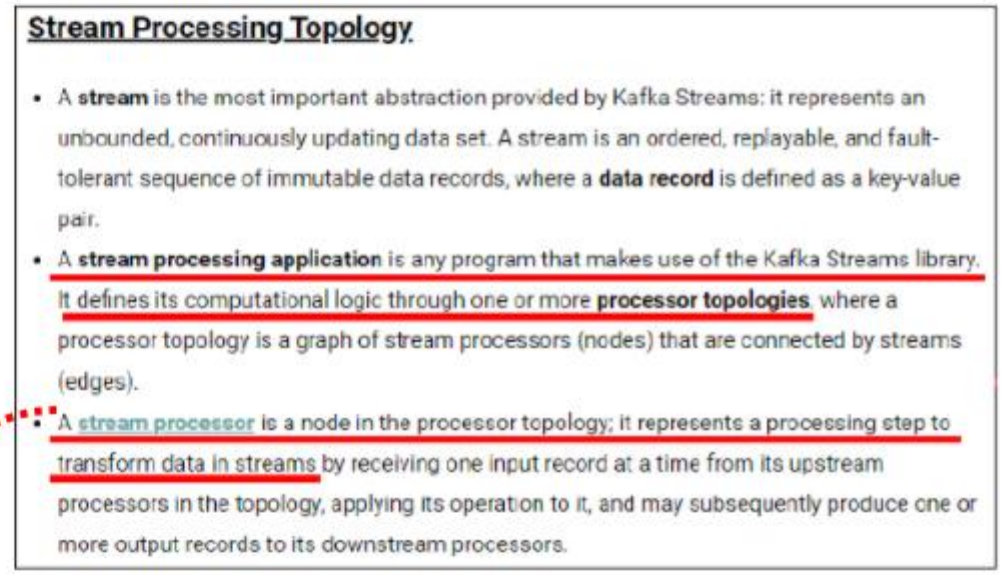


Source: https://kafka.apache.org/36/documentation/streams/architecture.

88. The '836 Accused Products determine "by a processor, that an application in an application sequence has failed during a communication session that is associated with the application sequence." For example, Kafka determines that a stream processor in a Kafka Streams application processor topology has failed while processing the stream. The stream is associated with the processor topology:

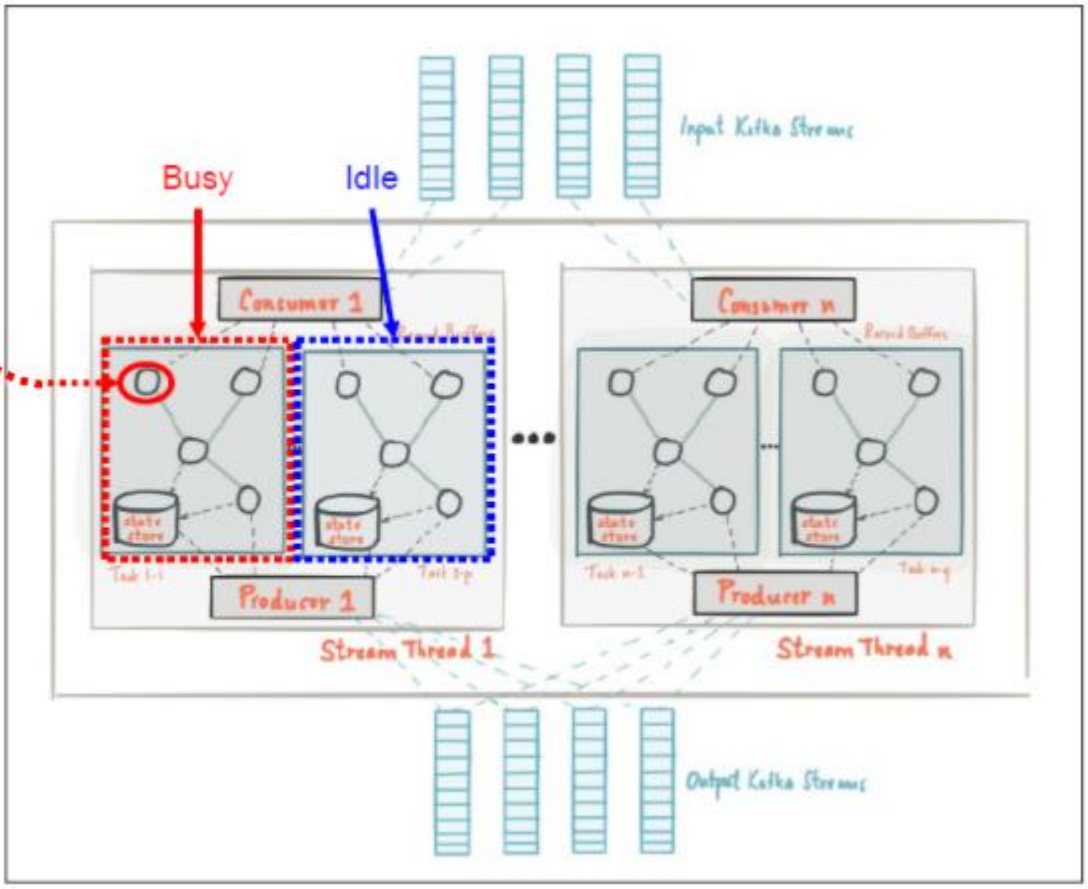

Slightly simplified, the maximum parallelism at which your application may run is bounded by the maximum number of stream tasks, which itself is determined by maximum number of partitions of the input topic(s) the application is reading from. For example, if your input topic has 5 partitions, then you can run up to 5 applications instances. These instances will collaboratively process the topic's data. If you run a larger number of app instances than partitions of the input topic, the "excess" app instances will launch but remain idle; however, if one of the busy instances goes down, one of the idle instances will resume the former's work.

Source: https://kafka.apache.org/36/documentation/streams/architecture #streams_architecture_tasks (emphasis added).



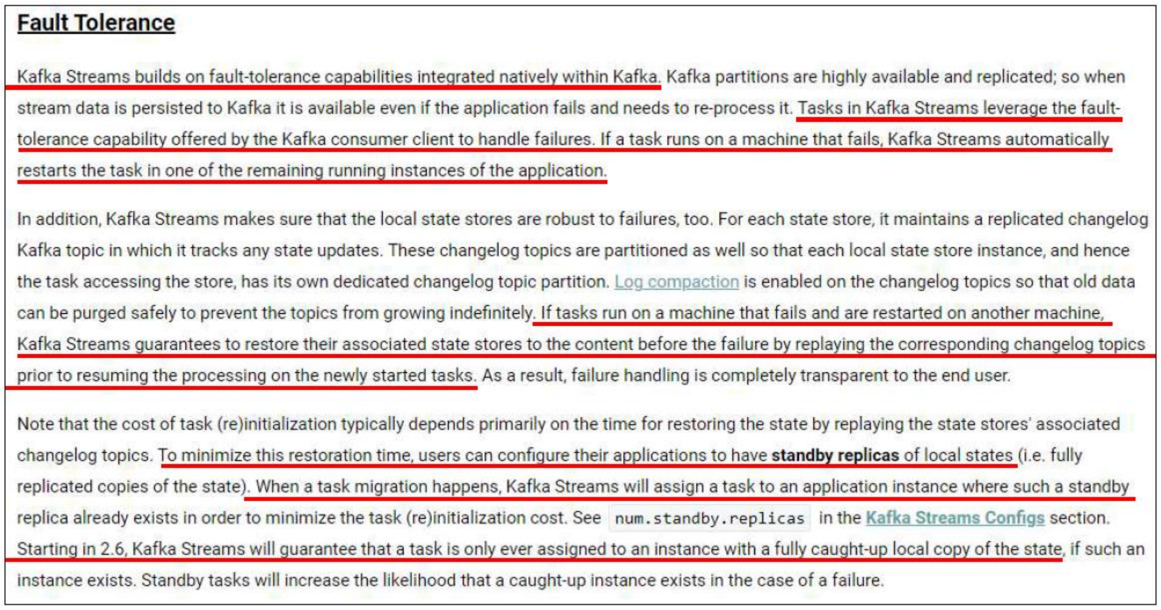
Stream Processing Topology

- A **stream** is the most important abstraction provided by Kafka Streams: it represents an unbounded, continuously updating data set. A stream is an ordered, replayable, and fault-tolerant sequence of immutable data records, where a **data record** is defined as a key-value pair.
- A **stream processing application** is any program that makes use of the Kafka Streams library. It defines its computational logic through one or more **processor topologies**, where a processor topology is a graph of stream processors (nodes) that are connected by streams (edges).
- A stream processor is a node in the processor topology; it represents a processing step to transform data in streams by receiving one input record at a time from its upstream processors in the topology, applying its operation to it, and may subsequently produce one or more output records to its downstream processors.



Sources: https://kafka.apache.org/36/documentation/streams/core-concepts#streams_topology; https://kafka.apache.org/36/documentation/streams/architecture (emphasis added).

89. The ’836 Accused Products send “by the processor, a re-establishment message to a replacement application, the re-establishment message referencing an identifier of at least one of the communication session and its dialog.” For example, Kafka sends a re-establishment message to an idle task instance, referencing an identifier of the stream (the stream identifier is referenced so that the idle instance may know which stream to process):



Fault Tolerance

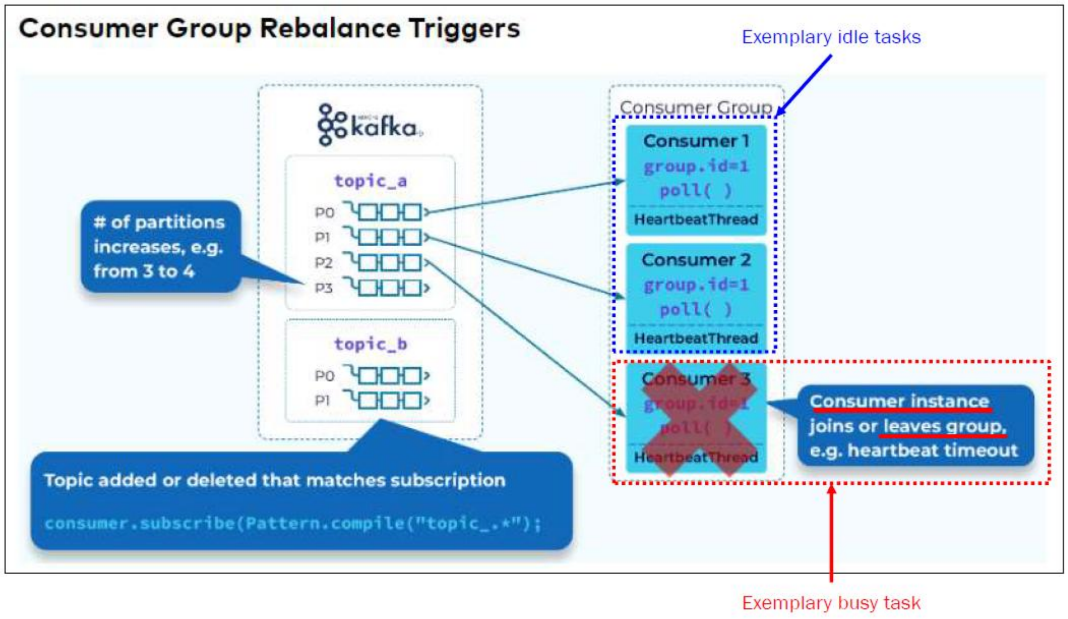
Kafka Streams builds on fault-tolerance capabilities integrated natively within Kafka. Kafka partitions are highly available and replicated; so when stream data is persisted to Kafka it is available even if the application fails and needs to re-process it. Tasks in Kafka Streams leverage the fault-tolerance capability offered by the Kafka consumer client to handle failures. If a task runs on a machine that fails, Kafka Streams automatically restarts the task in one of the remaining running instances of the application.

In addition, Kafka Streams makes sure that the local state stores are robust to failures, too. For each state store, it maintains a replicated changelog Kafka topic in which it tracks any state updates. These changelog topics are partitioned as well so that each local state store instance, and hence the task accessing the store, has its own dedicated changelog topic partition. Log compaction is enabled on the changelog topics so that old data can be purged safely to prevent the topics from growing indefinitely. If tasks run on a machine that fails and are restarted on another machine, Kafka Streams guarantees to restore their associated state stores to the content before the failure by replaying the corresponding changelog topics prior to resuming the processing on the newly started tasks. As a result, failure handling is completely transparent to the end user.

Note that the cost of task (re)initialization typically depends primarily on the time for restoring the state by replaying the state stores' associated changelog topics. To minimize this restoration time, users can configure their applications to have **standby replicas** of local states (i.e. fully replicated copies of the state). When a task migration happens, Kafka Streams will assign a task to an application instance where such a standby replica already exists in order to minimize the task (re)initialization cost. See `num.standby.replicas` in the Kafka Streams Configs section. Starting in 2.6, Kafka Streams will guarantee that a task is only ever assigned to an instance with a fully caught-up local copy of the state, if such an instance exists. Standby tasks will increase the likelihood that a caught-up instance exists in the case of a failure.

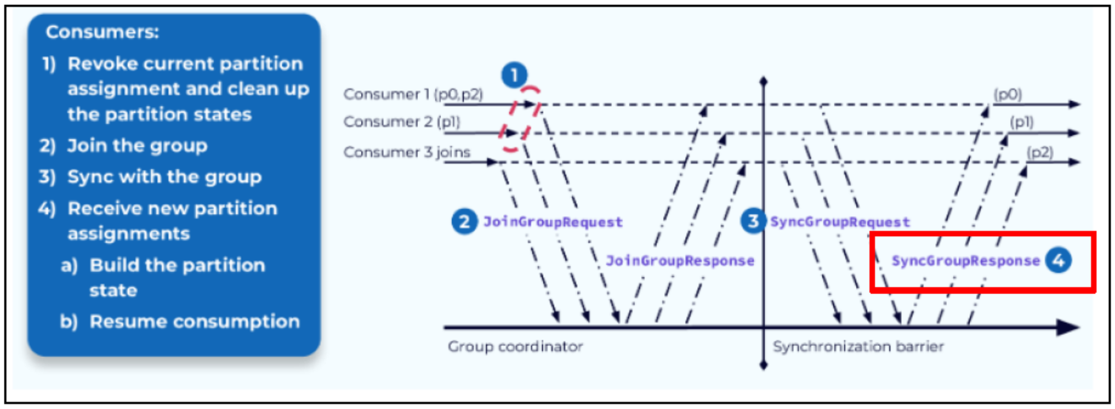
Source: https://kafka.apache.org/36/documentation/streams/architecture (emphasis added).

The consumer client fault tolerance (which is applicable to Kafka Streams Applications) behaves in accordance with Consumer Group Rebalance:

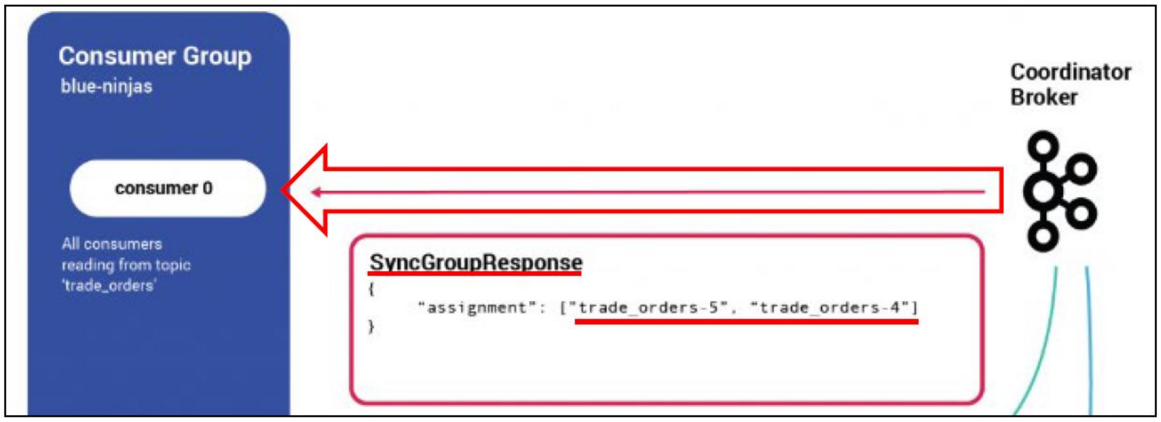

Consumer Group Rebalance Triggers
Exemplary idle tasks
kafka
topic_a
P0
P1
P2
P3
topic_b
P0
P1
# of partitions increases, e.g. from 3 to 4
Consumer Group
Consumer 1
group.id=1
poll( )
HeartbeatThread
Consumer 2
group.id=1
poll( )
HeartbeatThread
Consumer 3
group.id=1
poll( )
HeartbeatThread
Consumer instance joins or leaves group, e.g. heartbeat timeout
Topic added or deleted that matches subscription
consumer.subscribe(Pattern.compile("topic_.*");
Exemplary busy task

Source: https://developer.confluent.io/courses/architecture/consumer-group-protocol/ (emphasis added).

Kafka sends a SyncGroupResponse to the idle task instance. The SyncGroupResponse references the partitions that the failed stream processor was processing.



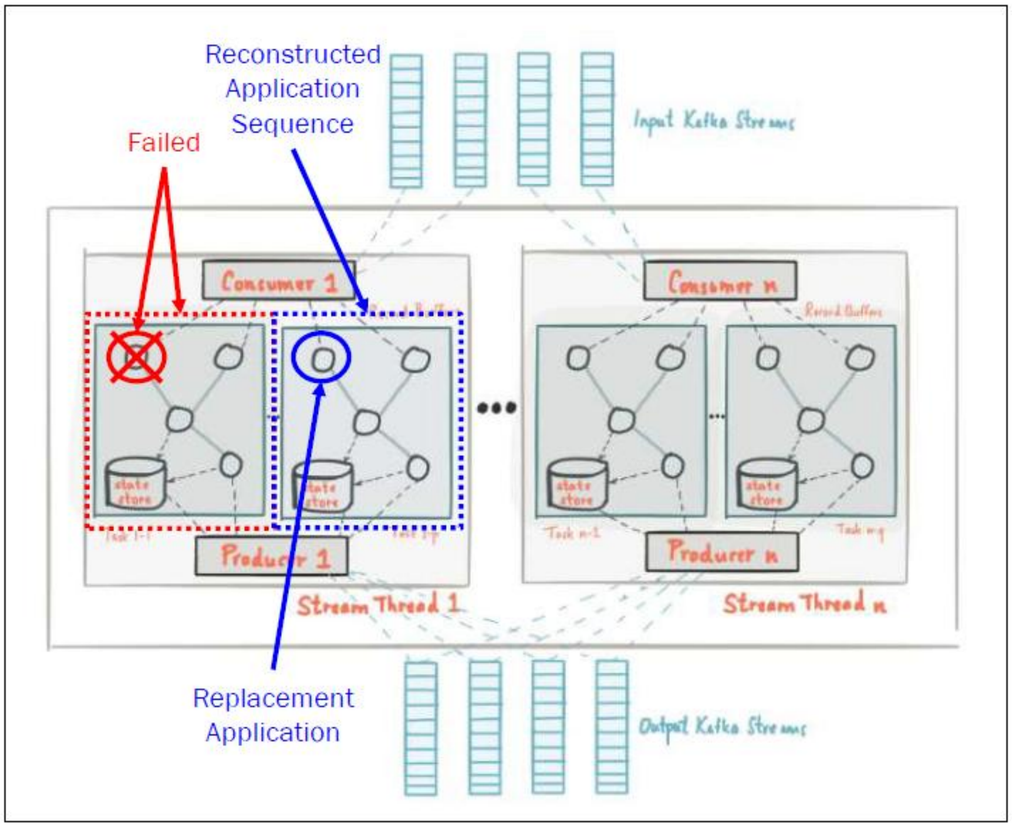
Source: https://developer.confluent.io/courses/architecture/consumer-group-protocol/ (emphasis added).



```
{
    "assignment": ["trade_orders-5", "trade_orders-4"]
}
```

Source: https://www.confluent.io/blog/apache-kafka-data-access-semantics-consumers-and-membership/ (emphasis added).

90. The ’836 Accused Products reconstruct “by the processor, the application sequence for the communication session while the communication session is still in progress so that the reconstructed application sequence includes the replacement application.” For example, Kafka reconstructs the processor topology for the stream so that the new instance of the task includes the replacement stream processor and can continue processing forward from the partition state:



Source: https://kafka.apache.org/36/documentation/streams/architecture (emphasis added).

91. The technology discussion above and the exemplary ’836 Accused Products provide context for Plaintiff’s infringement allegations.

92. At a minimum, Defendant has known of the ’836 Patent at least as early as the filing date of the Complaint. In addition, Defendant has known about the ’836 Patent since at least March 8, 2024, when Plaintiff sent correspondence to Defendant via FedEx alerting Defendant to its infringement.

93. On information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has actively induced, under U.S.C. § 271(b), its distributors, customers, subsidiaries, importers, and/or consumers that import, purchase, or sell the ’836 Accused Products that include or are made using all of the limitations of one or more claims of the ’836 Patent to directly infringe one or more claims of the ’836 Patent (e.g., claim 1, as

discussed above) by using, offering for sale, selling, and/or importing the ’836 Accused Products. Since at least the notice provided on the above-mentioned date, Defendant does so with knowledge, or with willful blindness of the fact, that the induced acts constitute infringement of the ’836 Patent. Defendant intends to cause, and has taken affirmative steps to induce infringement by its distributors, importers, customers, subsidiaries, and/or consumers by at least, *inter alia*, creating advertisements that promote the infringing use of the ’836 Accused Products, creating and/or maintaining established distribution channels for the ’836 Accused Products into and within the United States, manufacturing the ’836 Accused Products in conformity with U.S. laws and regulations, distributing or making available instructions or manuals for these products to purchasers and prospective buyers, and testing the ’836 Accused Products, and/or providing technical support, replacement parts, or services for these products to these purchasers in the United States. For example, Defendant configures Apache Kafka in the ’836 Accused Products to contain specific instructions, such as source code and configuration files, that cause such products to support at least its Webinar and RingSense products.

94. In the alternative, on information and belief, since at least the above-mentioned date when Defendant was on notice of its infringement, Defendant has contributorily infringed, under U.S.C. § 271(c), one or more claims of the ’836 Patent. For example, Defendant contributes to the direct infringement of such claims by distributors, customers, subsidiaries, importers, and/or consumers that use, import, purchase, or sell the ’836 Accused Products. To the extent that the ’836 Accused Products do not directly infringe one or more claims of the ’836 Patent, such products contain instructions, such as source code, that are especially adapted to cause the ’836 Accused Products to operate in an infringing manner. Such instructions are specifically designed to cause the ’836 Accused Products to provide and utilize Apache Kafka in an infringing manner

and are a material part of the invention of the ’836 patent and are not a staple article of commerce suitable for substantial non-infringing use.

95. On information and belief, despite having knowledge of the ’836 Patent and knowledge that it is directly and/or indirectly infringing one or more claims of the ’836 Patent, Defendant has nevertheless continued its infringing conduct and disregarded an objectively high likelihood of infringement. Defendant’s infringing activities relative to the ’836 Patent have been, and continue to be, willful, wanton, malicious, in bad-faith, deliberate, consciously wrongful, flagrant, and an egregious case of misconduct beyond typical infringement such that Plaintiff is entitled under 35 U.S.C. § 284 to enhanced damages up to three times the amount found or assessed.

96. Plaintiff has been damaged as a result of Defendant’s infringing conduct described in this Count. Defendant is, thus, liable to Plaintiff in an amount that adequately compensates Plaintiff for Defendant’s infringements, which, by law, cannot be less than a reasonable royalty, together with interest and costs as fixed by this Court under 35 U.S.C. § 284.

CONCLUSION

97. Plaintiff is entitled to recover from Defendant the damages sustained by Plaintiff as a result of Defendant’s wrongful acts, and willful infringement, in an amount subject to proof at trial, which, by law, cannot be less than a reasonable royalty, together with interest and costs as fixed by this Court.

98. Plaintiff has incurred and will incur attorneys’ fees, costs, and expenses in the prosecution of this action. The circumstances of this dispute may give rise to an exceptional case within the meaning of 35 U.S.C. § 285, and Plaintiff is entitled to recover its reasonable and necessary attorneys’ fees, costs, and expenses.

JURY DEMAND

Plaintiff hereby requests a trial by jury pursuant to Rule 38 of the Federal Rules of Civil Procedure.

PRAYER FOR RELIEF

Plaintiff respectfully requests that the Court find in its favor and against Defendant, and that the Court grant Plaintiff the following relief:

1. A judgment that Defendant has infringed the Asserted Patents as alleged herein, directly and/or indirectly by way of inducing infringement of such patents;
2. A judgment for an accounting of all damages sustained by Plaintiff as a result of the acts of infringement by Defendant;
3. A judgment and order requiring Defendant to pay Plaintiff damages under 35 U.S.C. § 284, including up to treble damages as provided by 35 U.S.C. § 284, and any royalties determined to be appropriate;
4. A judgment and order requiring Defendant to pay Plaintiff pre-judgment and post-judgment interest on the damages awarded;
5. A judgment and order finding this to be an exceptional case and requiring Defendant to pay the costs of this action (including all disbursements) and attorneys' fees as provided by 35 U.S.C. § 285; and
6. Such other and further relief as the Court deems just and equitable.

Dated: May 16, 2025

OF COUNSEL:

Patrick J. Conroy
Justin B. Kimble
Jon Rastegar
Nathan L. Levenson
NELSON BUMGARDNER CONROY PC
2727 N. Harwood St., Suite 250
Dallas, Texas 75201
(817) 377-9111
pat@nelbum.com
justin@nelbum.com
jon@nelbum.com
nathan@nelbum.com

BAYARD, P.A.

/s/ Stephen B. Brauerman
Stephen B. Brauerman (#4952)
Ronald P. Golden III (#6254)
600 N. King Street, Suite 400
P.O. Box 25130
Wilmington, Delaware 19801
(302) 655-5000
sbrauerman@bayardlaw.com
rgolden@bayardlaw.com

Attorneys for Plaintiff
Arlington Technologies LLC